DISTRIBUTION LIST

B1800/B1700 SOFTWARE PRODUCT SPECIFICATIONS

DETROIT

J. Garren - Prod. Mgmt.
P. Gonzales - Prod. Mgmt.
J. M. Ross - Int'l Group P
C. Kunkelmann - BMG
J. McClintock - CSG
D. Dahm - Corp. Eng.
Dir., Pgmg. - SSG
M. Dowers - Int'l FE
D. Hill - TC, BM, & SS

U.S. AND EUROPE

D. Cikoski - (Plymouth)
J. H. Pedersen (Plymouth)
W. E. Feeser (Austin)
J. Berta (Downingtown)
W. Minarcik (Paoli)
G. Smolnik (Paoli)
M. E. Ryan (Tredyffrin)
T. Yama - F&SSG (McLean)
J. Poterack - F&SSG (McLean)
A. Kosla - F&SSG (McLean)
A. LaCivita - F&SSG (McLean)
L. Guell - F&SSG (McLean)
R. Sutton - F&SSG (McLean)
L. DeBartelo - WADC (Irvine)
R. Cole (Pasadena)
H. M. Townsend (Pasadena)
N. Cass - Pat. Atty. (Pasadena)
S. Samman (Mission Viejo)
J. Lowe (Mission Viejo)
H. N. Riley (El Monte)
J. C. Allan (Glenrothes)
W. McKee (Cumbernauld)
B. Higgins (Livingston)
Mgr, NPSGrp (Ruislip)
E. Norton (Middlesex)
J. Gerain (Pantin)
J. Cazanove (Villers)
J. C. Wery (Liege)
R. Bouvier (Liege)
G. LeBlanc (Liege)
C. J. Tooth - SSG (London)
J. Dreystadt (Wayne)

SANTA BARBARA PLANT

R. Shobe
K. Meyers
R. Bauerle
E. Yardi
A. van der Linden - 12

Distribution list current as of 10/22/81

Burroughs Corporation

COMPUTER SYSTEMS GROUP
SANTA BARBARA PLANT

2228 3543

BOLT EXECUTIVE

PRODUCT SPECIFICATION

REV LTR	REVISION ISSUE DATE	APPROVED BY	REVISIONS
A	1/8/82	RS Wohl	Original Issue -- Special Mark 10.0 Release

BURROUGHS CORPORATION
COMPUTER SYSTEMS GROUP
SANTA BARBARA PLANT

COMPANY CONFIDENTIAL
B1000 BOLT EXECUTIVE
P.S. 2228 3543(A)

TABLE OF CONTENTS

BURROUGHS CORPORATION
COMPUTER SYSTEMS GROUP
SANTA BARBARA PLANT

INTRODUCTION

BOLT is an acronym for B1000 On-Line Tests. BOLT is a normal state I/O test program that determines and isolates the failure mode of a peripheral and provides a level of confidence in the peripheral after any repairs have been made. BOLT only implicitly tests the I/O controls (i.e., if the peripheral performs correctly then the control is implicitly functional). I/O testing is accomplished either through pre-defined test sections or through BOLT's Peripheral and Control Analysis Program (PCAP) feature which allows the operator to build and dispatch up to six (6) descriptors to an I/O device.

BOLT is structured as a single SOL program with one executive and a separate procedure for each I/O device supported. BOLT is segmented in such a way that only the code necessary to perform a specific function needs to be present in memory.

The executive acts as the interface between the operator and the test sections and handles all solicited and unsolicited ODT input. The executive schedules and executes test sections, builds and dispatches PCAP descriptors and displays error and status messages.

BOLT allows only one device to be under test at a time. Multiple devices may be tested at the same time by executing multiple copies of BOLT.

RELATED DOCUMENTS

Part Number	Part Description
P.S. 2228 3600	B1000 BOLT Multiline Control Test
P.S. 2228 3592	B1000 BOLT Magnetic Tape Test
P.S. 2228 3618	B1000 BOLT Line Printer Test
P.S. 2228 3626	B1000 BOLT Card Device Test
P.S. 2212 5421	B1000 DC/AUDIT

DEFINITIONS

Following are brief definitions for some of the terms and phrases used in this document.

BOLT

Acronym for B1000 On-Line Tests. Also the name of the object code file that must be executed to run the executive and test programs.

COMMAND

Any valid instruction that the BOLT executive can parse. Usually entered by the operator through the ODT.

CONFIDENCE TEST

Any test that attempts to operate a peripheral and control in a normal functional manner.

CONTROL

One or more logic cards that interface a peripheral to a processor.

DEFAULT

Refers to test parameters or test sections that are automatically used by BOLT unless otherwise specified by the operator.

DEVICE

A peripheral connected to an I/O interface control.

DIAGNOSTIC TEST

Any test that uses diagnostic operation codes to attempt to detect and isolate control errors to an interface card or functional logic block.

OPERATOR

The person that controls the operation of BOLT through input commands.

OPERATOR SENSITIVE

Refers to test sections whose default parameters may be changed by the operator.

PCAP

Acronym for Peripheral and Control Analysis Program -- a feature of BOLT that allows the operator to build and dispatch up to six descriptors.

SECTION

A functionally complete subdivision of a test procedure that may be individually scheduled and executed by the operator.

SYNTAX ERROR

An error in the BOLT commands entered by the operator.

TEST

Usually refers to the complete collection of sections for a certain type of peripheral such as the Line Printer TEST. May also refer to one section or a group of PCAP descriptors built by the operator to exercise a peripheral.

UNIT

In most cases equivalent with Device. In BOLT, UNIT is a command which is used to gain access to a device or its entire control. A device is usually selected using an MCP mnemonic such as LPA for Line Printer A.

DESIGN GOALS

The following goals have been applied to the design of the executive and test routines:

1. There is one executive which parses all commands and contains the procedures that are common to all tests. The executive and the test routines consist of one SDL program, where the tests are written as separate procedures that can be added to the source file.

2. BOLT is able to be efficiently multi-programmed in 128KB systems. This means that the run time memory requirements for BOLT should never exceed 60KB.

3. The tests do not require that the system be halted or clear/started.

4. Whenever possible, confidence test sections are designed in a bottom-up fashion in which the first section executed performs the simplest operations using the least amount of logic. Subsequent sections rely on previously verified logic as much as possible. The optional diagnostic sections are designed in a similar manner.

5. All device tests use a special diagnostic communicate to the MCP, which inhibits the MCP from reporting errors, retrying ops, or handling exceptions when sending the descriptor to the device.

6. Program input and output messages have a minimum impact on the ODT so as not to interfere with other On-Line programs communicating with the ODT. Output is optional to the line printer or ODT. BOLT is sensitive to non-solicited ODT input to alter program execution.

7. All tests insure that they do not cause data corruption on other peripherals that are being used by the system while a device or interface is being tested.

8. Program error and operation messages are maintained in separate files. This will allow easy alterations for nationalistic language requirements.

COMMAND SYNTAX

BOLT commands may be entered in upper case or lower case (lower case commands are converted to upper case by the MCP). Blanks or commas may be used to separate statement components. The number of blanks and commas is not significant. Only the first three letters of any command are significant for each identifier. More letters may be entered for clarity.

If a syntax error is encountered during the scanning of a statement, a short error message will appear on the ODT. The input line up to the point of error will be displayed to help isolate the syntax error. The remainder of the unscanned input is discarded. All commands parsed correctly up to the point of error are executed before the error message is displayed.

Numbers may be entered in either decimal or hexadecimal. An at sign "@" preceding and following a number denotes a hexadecimal value. This is the case for numbers displayed on the ODT and numbers entered by the operator.

It is valid to transmit a blank character to the test program. This will cause the program to suspend execution of the current section (if one is being executed) and wait for a GO command. Entering a blank is a quick way to cause a test to pause and accept new parameters.

**** NOTE ****

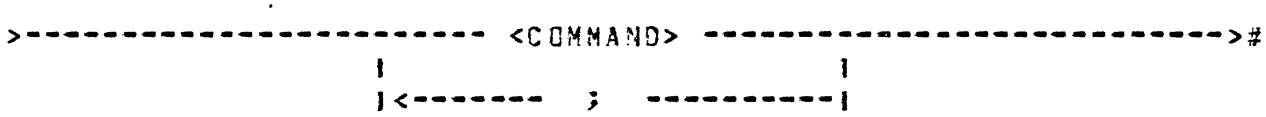
Bolt parses and executes commands until a GO, SEND, Logical Op (WRITE, TEST, etc), EXIT, CLEAR or RESTART command is entered. **Any input following one of these commands on the same line will be ignored by BOLT.** For example:

```
<job #> AX UNIT MTA; SEC 1, 2;GO;SEC 3, 4;GO
```

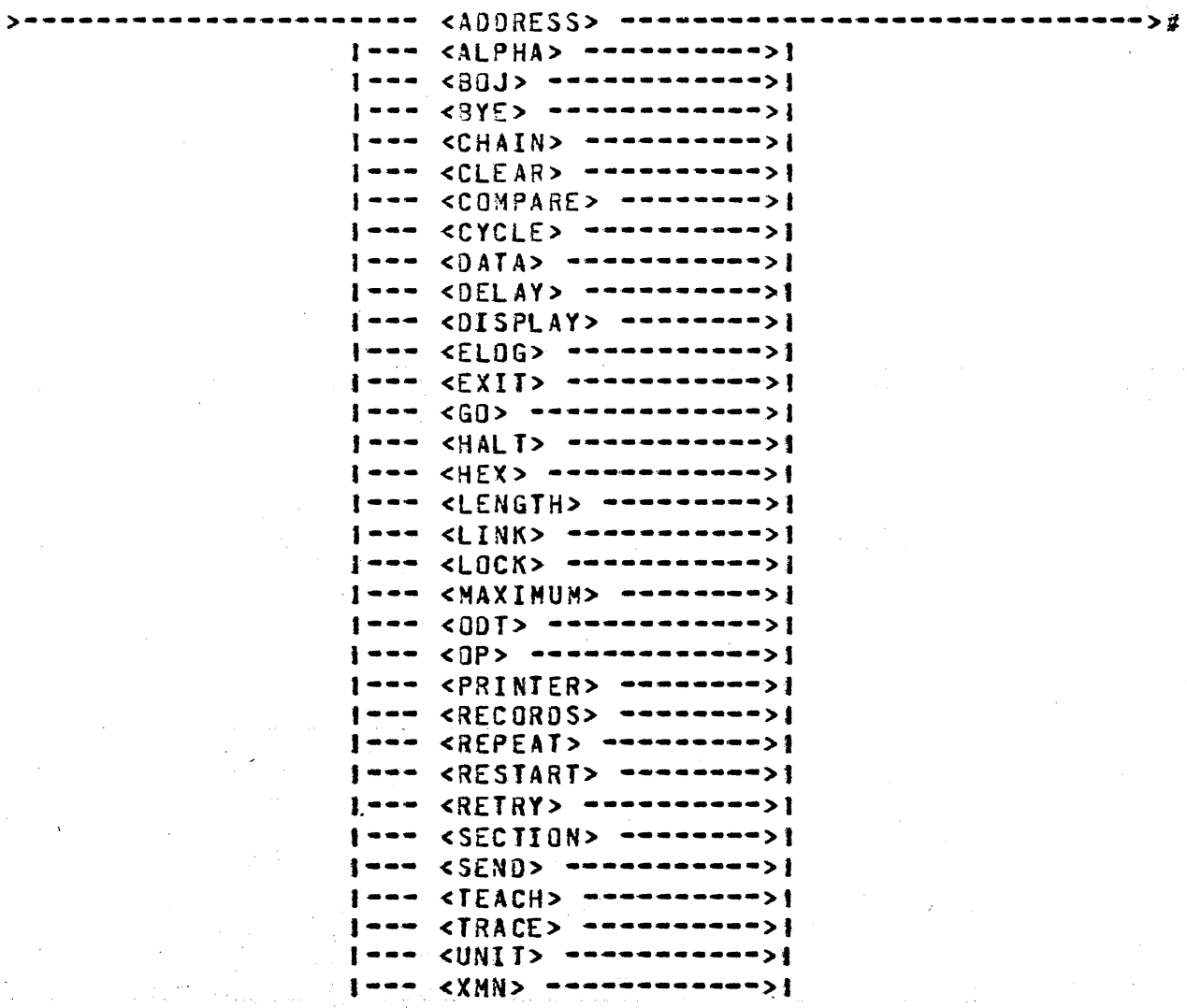
would be incorrect. The correct entry would be:

```
<job #> AX UNIT MTA; SEC 1, 2;GO

(wait until section 2 completes)

<job #> AX SEC 3, 4;GO
```

STATEMENT



COMMAND

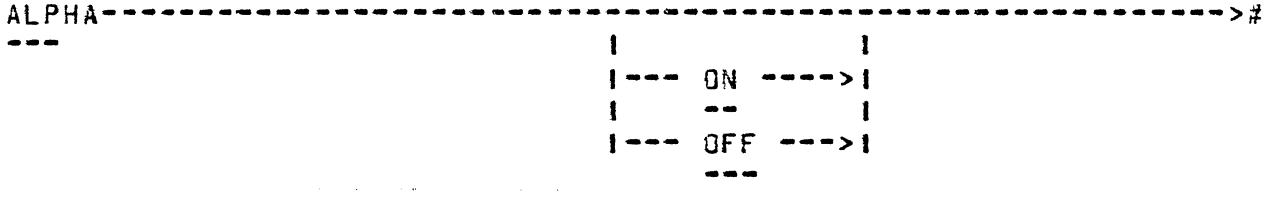


```
ADDRESS---------- " --- <2-EBCDIC-characters> --- " ---------->#
---
```

MULTILINE DEVICES ONLY

The ADDRESS command scans off the two EBCDIC characters enclosed in quotes and stores them in a variable called TERMINAL_ADDRESS which is used by predefined datacomm tests to determine which terminal to select.

Examples:	ADDRESS "D2"	The next data comm test that requires a terminal address will use "D2".
	ADD "D3"	Use "D3" for terminal address.



The ALPHA command permits data to be displayed in EBCDIC (character) format.

If this option is set when data is being displayed, the data will be displayed in EBCDIC format.

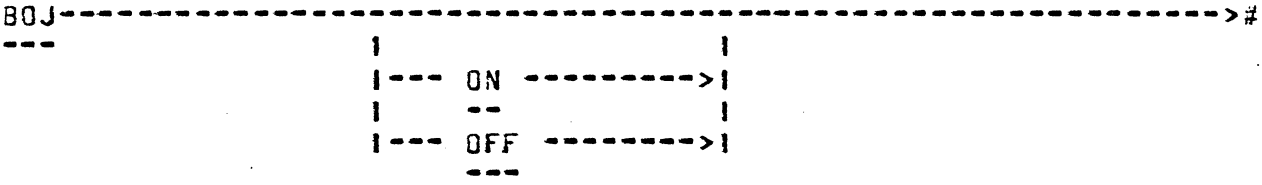
This command is independent of the HEX command.

It should be noted that this command does not display the data. It only controls the format of the data when it is displayed.

If ALPHA is entered without the modifiers ON or OFF the option will be set on.

Default value: OFF

Examples:		
	ALP OFF	Data will not be displayed in EBCDIC format.
	ALPHA	Data will be displayed in EBCDIC format.



The BOJ command causes "Beginning of Job" and "Ending of Job" messages to be displayed at the start and end of all selected test sections.

If BOJ is entered without the modifiers ON or OFF the option will be set on.

This command is useful when minimum number of display messages are desired while REPEAT is on.

Default value: ON

Examples:		
	BOJ OFF	BOJ and EOJ messages will not be displayed.
	BOJ	BOJ and EOJ messages will be displayed.

BYE---------->#

The BYE command causes BOLT to terminate. The program will go to "END OF JOB", then be removed from the MCP job mix. If there is any device currently in use by BOLT then an attempt is made to restore the device back to its normal operating condition and release it back to MCP control. If the attempt fails then an error message is displayed telling why and the BYE command will have no effect. Otherwise, BOLT will go to EOJ as intended.

Example: BYE

CHAIN ---------- <NUMBER> ----------#

PRINTER DEVICES ONLY

The CHAIN command is for use with the On-Line Printer Test only. To run some Printer Test Sections a default data pattern must first have been selected. Some Printers aid in the selection of this default data by providing a chain id in the result status field of a Test op. If the Line Printer Test cannot determine the proper data pattern to select from information returned by the Test op, the BOLT operator will be asked to select and enter a chain id.

This command may also be used to override the data pattern selected by the Line Printer Test. Such an action would be necessary when the Test has made a wrong assumption about the chain id returned by a Test op.

When asked to select and enter a chain id, the operator will be shown a list of chain ids to select from. The chain id entered must be one from that list. At present, the Line Printer Test supports the following character sets.

```
 1 =  16 CHARACTER EBCDIC
 2 =  18 CHARACTER EBCDIC
 3 =  48 CHARACTER EBCDIC
 4 =  48 CHARACTER FORTRAN (Standard)
 5 =  48 CHARACTER FORTRAN (Non-Standard)
 6 =  48 CHARACTER B500
 7 =  48 CHARACTER RPG
 8 =  64 CHARACTER EBCDIC OR BCL
 9 =  72 CHARACTER EBCDIC
10 =  72 CHARACTER ASCII
11 =  72 CHARACTER ALPHABETIZED
12 =  72 CHARACTER NUMBERIZED
13 =  96 CHARACTER EBCDIC
14 = 144 CHARACTER THIALAND
```

Examples:

```
CHAIN 1     Select the EBCDIC 16 character set.
CHAIN 13    Select the EBCDIC 96 character set.
```

CLEAR---------->#

The CLEAR command causes the test section that is currently executing to be terminated. The test program will return to the idle state.

All options and variables will be reset to their default values. Any remaining test sections will not be executed. The unit selected to be tested will not be changed, but its default section list will be reloaded.

Example: CLEAR

****** NOTE ******

Any input following this command on the same line will be ignored by BOLT.

COMPARE-------- <POINT> ------- <POINT> ---------->#

The COMPARE command causes DATA areas of PCAP descriptors to be compared.

The contents of the DATA area designated by the first <POINT> value is compared with that of the DATA area designated by the second <POINT> value. The number of bytes compared are specified by the LENGTH value for the second (standard) descriptor data area.

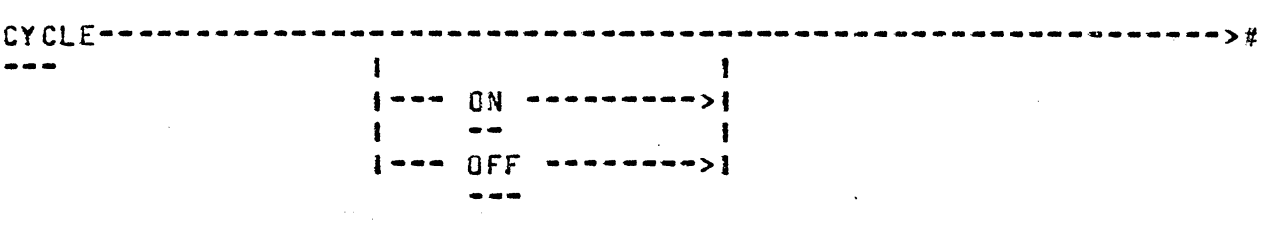
If the HEX option is set, the data will be compared in four bit quantities and the decimal position of the first "hex digit" is displayed at the beginning of each line in parenthesis. If the ALPHA option is set, the data will be compared in eight bit quantities and the decimal position of the first "character" is displayed at the beginning of each line in parenthesis. If both the ALPHA and HEX are selected the data will be compared in four bit quantities.

If the ODT option is set then 46 characters of "EXPECTED" and "OBSERVED" data will be displayed to the ODT beginning at the character boundary of the data in error. If the PRINTER option is set then the entire data buffer is displayed to the printer.

Compare errors are marked with asterisks (*) on the line titled "ERRORS" for each ALPHA character or HEX digit in error.

Example: COM [4] [2] The contents of DATA [4] will be compared with that of DATA [2] for LENGTH [4] bytes.

```
EXPECTED (00011): COMPARE EXAMPLE. THIS IS A COMPARE EXAMPLE.
OBSERVED        : BOMPARE ECAMPLE. THIS IS A BOMPARE ECAMPLE.
ERRORS          : *       *                *       *
```



The sequence of sections specified in the SECTION command will be continuously repeated if the CYCLE command is set.

If CYCLE is either followed by ON or not followed by a modifier, the selected sequence of sections will be repeated until a RESTART, EXIT, CLEAR, BYE or CYCLE OFF statement is entered.

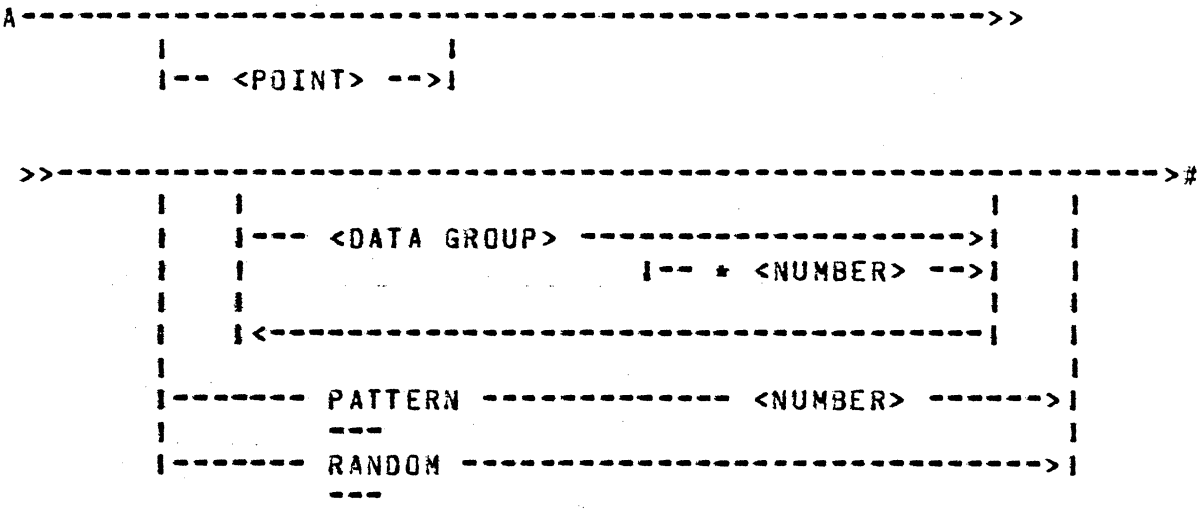
Default value: OFF

Example:	CYCLE OFF	Execute the sequence of sections once or stop the cycle process.
	CYCLE	Execute the sequence of sections until a STOP, RESTART, CYCLE OFF or BYE command is entered.

The following two examples illustrate the difference between the CYCLE and REPEAT commands.

REPEAT 3; SECTION 1,3; will cause sections to be executed in the following order: 1,1,1,3,3,3.

CYCLE ON; SECTION 1,3; will cause sections to be executed in the following order: 1,3,1,3,1,3,1,3,.....



The DATA command is used to select data patterns to be used by a test section or by a PCAP descriptor.

The <POINT> modifier is used to indicate that this command applies to a specific PCAP descriptor. If <POINT> is not used then the data is used by the current test section if it is sensitive to that parameter.

If DATA is not followed by a data group or modifing word, the data area is filled with hex zeros.

If fewer bytes of data are entered than the data area can contain, the test program will fill the remainder of the data area with a repeating pattern of the data that was entered.

Data may be entered in hexadecimal, character data, or constant data groups. The data groups will be concatenated and stored as a single data string. If a data group is followed by an asterisk and a number, the data group will be entered into the data string the designated number of times. Hexadecimal data must be entered in groups of six hex digits or less.

Only the amount of data specified by the LENGTH command will be used. The PCAP data buffers have a maximum size of 1024 bytes each. Test section data is entered using the DATA command with no <POINT> modifier. In this case the data entered by the operator is stored in a 1024 byte buffer called DATA_ARRAY. If a test section is data sensitive, then the section will copy the data from DATA_ARRAY into its own test buffer before the section is begun.

For example, the following commands would cause the data array to be filled as shown. DATA @23@,@AB@,"F";

Data array = @23ABC623ABC623ABC6.....23AB@

If the modifier RANDOM is used then the selected data buffer will be filled with a repeating 256 byte psuedo-random hex string.

The data buffers may be filled with a repeating copy of a predefined data pattern by entering the PATTERN modifier followed by the desired pattern numoer. The following patterns are supported by BOLT:

PATTERN 4 - EBCDIC sixteen character set
PATTERN 5 - EBCDIC forty-eight character set
PATTERN 6 - EBCDIC sixty-four character set
PATTERN 7 - EBCDIC seventy-two character set
PATTERN 8 - EBCDIC ninty-six character set
PATTERN 9 - FORTRAN forty-eight character set
PATTERN 10 - B500 forty-eight character set
PATTERN 11 - RPG forty-eight character set
PATTERN 13 - 0 to 255 incremented by 1

All other pattern numbers are invalid and will return an error message if entered.

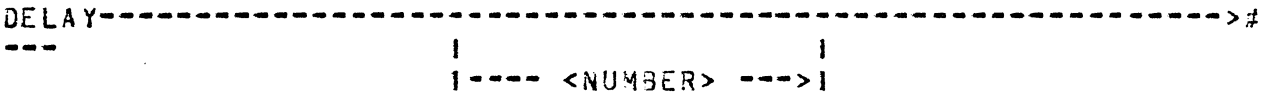
The following data constants are supported by BOLT and their use in the DATA command will cause the corresponding EBCDIC code to be stored in the DATA array:

EOT	NUL	SOH	STX	ETX	FSL
BSL	ENQ	ACK	BEL	NAK	LF
POL	SEL	FF	CR	ESC	DEL
DLE	SUB	CAN	EM	ETB	DC1
DC2	DC3	DC4			

Default value: DATA

Examples:

DATA	The default data will be used for the next test section run.
DATA [1]	The DATA array for PCAP descriptor [1] is filled with hex zeros.
DATA [5] @12@	The DATA array for PCAP descriptor [5] is filled with a repeating pattern of @12@.
DATA @3@, "X" * 3, "5"	The DATA array for the next test section will be filled with the repeating pattern @3E7E7E7F5@.
DATA RANDOM	The test section data array will be filled with a pseudo-random data pattern.
DATA [0] PAT 4	The DATA array for PCAP descriptor [0] will data pattern four.
DATA SOH "02" STX "TEST" DC1 ETX	Build a standard select message in the DATA array for use in datacomm testing.



The DELAY command causes most IO's to delay <NUMBER> * 0.1 seconds before checking for a timeout condition. <NUMBER> may be specified in decimal or hex and represents increments of one tenth of a second.

If DELAY is entered with no parameter then it is reset to its default value of 100 (i.e. 10 seconds).

Any test section may override the default or operator specified delay value for a particular IO, but it will be immediately restored when the IO is finished.

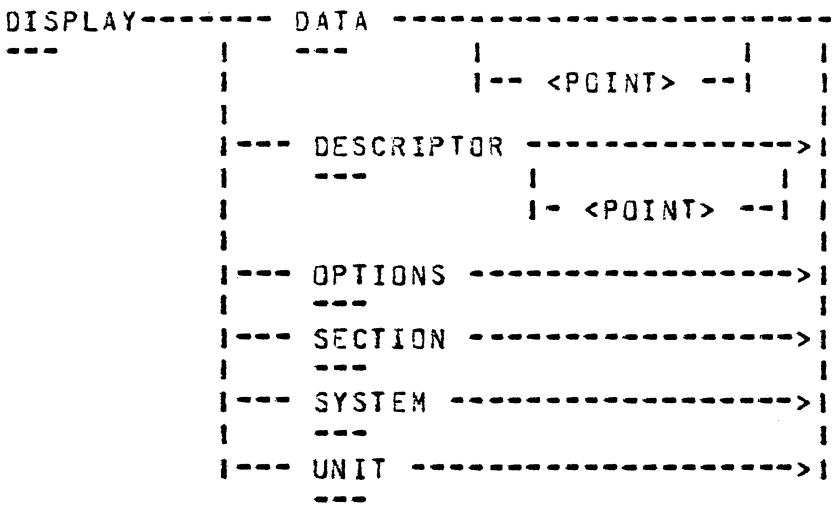
Default value: 100 (10 seconds)

Minimum value: 1 (0.1 seconds)

Maximum value: 3,000 (5 minutes)

Examples:

DELAY 600	Set timeout delay value to 60 seconds.
DELAY	Restore timeout delay to default of 10 seconds.



The DISPLAY DESCRIPTOR command displays the PCAP descriptor specified by the index. If no index is provided then it displays the TEST_DESC currently being used by the test section.

In the example shown below, the LINK field displays the index, [#], of the descriptor linked to. A value of -000001 indicates no link is made. The LENGTH field is the number of bytes in HEX (DECIMAL) specified by the LENGTH and DATA commands ((B_FIELD - A_FIELD)/8). For example:

```
53AX DISP DESC [0]
                        DESCRIPTOR 0
                        ------------

     OP:        400001   A_FIELD:  009BDF   RESULT:    800080
     LINK:   -0000001    B_FIELD:  00A3DF   XRSLT1:  00000000
     PRT/CH:       7C    C_FIELD:  000000   XRSLT2:  00000000
                          LENGTH:  000100 (000256)
```

The DISPLAY OPTIONS command displays the current state of all of the operator selectable commands which can be turned ON or OFF or can be assigned a numerical value. Numerical values are displayed in decimal.

In the example below TEST STATUS indicates whether a test section has been halted before completion with ACTIVE, while IDLE indicates that no section is running.

SECTION NUMBER is the current section running or the next section to be run. REPEAT VALUE is the value selected using the REPEAT command. REPEAT COUNT is set to REPEAT value and decremented once each time through a repeat loop. REPEAT COUNT therefore indicates the number of time left to repeat before completion.

RETRY VALUE is set by the RETRY command. NUMBER OF RECORDS is set by the RECORDS command.

SECTION ERRORS is the number of errors recorded for the most recent section. UNIT ERROR COUNT is the total number of errors accumulated for all sections run on that unit. MAXIMUM UNIT ERRORS is set by the MAXIMUM command and indicates the total UNIT error count for which the test will terminate.

DATA LENGTH is set by the LENGTH command and will read "DEFAULT" when a new unit is selected or a CLEAR or LEN command is entered. This informs the test sections to use predefined default lengths. If the operator specifies a value using the LENGTH command then the specified value will be displayed as DATA LENGTH and if a test section is "operator sensitive" then the new length value will be used.

DELAY TIMEOUT is set by the DELAY command and specifies the amount of time in tenths of seconds that BOLT will wait before reporting soft timeouts on all IO's.

TERMINAL ADDRESS is set by the ADDRESS command. TRANSMISSION number is set by the XMN command. These variables are used by the MULTILINE.

TEST SECTION DATA will display DEFAULT if a UNIT, CLEAR, RESTART or DATA command is entered which means the test section will use a predefined default data pattern. If the operator specifies a value for the DATA command, then the message "USER SPECIFIED" will be displayed in this field and those test sections that are "operator sensitive" to data will use the new data. Following is an example of the input and output for the DISPLAY OPTIONS command:

```
17AXDATA 3140;LEN 16;RECORDS 4;DISP OPT

                        BOLT OPTIONS
                        ------------

   TEST STATUS:..... IDLE      ADAPTER SELECTED:... NO
   UNIT SELECTED:.... MTA      CONTROL SELECTED:... YES
   SECTION NUMBER:... 012      NUMBER OF RECORDS:.. 00000004
   REPEAT VALUE:..... 000020   DATA LENGTH:........ 00000016
   REPEAT COUNT:..... 000000   TEST SECTION DATA:.. USER SPECIFIED
   RETRY VALUE:...... 000004   DELAY TIMEOUT:...... 0000100
   UNIT ERROR COUNT:. 000000   MAXIMUM UNIT ERRORS: 0000000
   SECTION ERRORS:... 000000   TRANSMISSION NUMBER:
   TERMINAL ADDRESS:.

   BOOLEAN OPTIONS:
      DISPLAY:    ODT =1   PRT =1  BOJ =0   HEX =0   ALPH =1
      ERROR:      LOCK=1   ELOG=1  HAL ERR=0
      TRACE:      BUF =0   DESC=0  AUDIT=0
      OTHER:      CYCL=1   HALT(I/O)=1
```

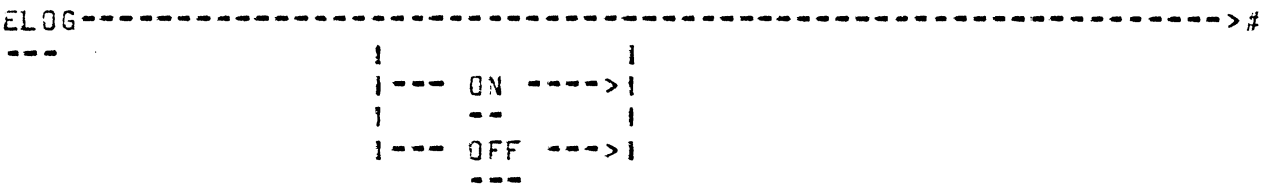
The DISPLAY DATA command will disolay the contents of the data array for the PCAP descriptor selected by <POINI>. If <POINI> is not provided, then the data area of the current test section will be displayed. If the ODT option is set the first 256 bytes of the buffer will be displayed to the ODT. If the PRINTER option is set then the entire data buffer is displayed to the printer. If the HEX option is set then the buffer will be displayed in hexadecimal. If the ALPHA option is set then the buffer will be displayed in ALPHA.

The DISPLAY SECTION command will display the current section list.

The DISPLAY SYSTEM command will display a list of devices connected to the system, along with port, channel and unit information.

The DISPLAY UNIT command will display the device that has been selected to to be tested by means of the UNIT command.

Examples:		
	DIS OPTIONS	The state of all of the operator selectable options will be displayed.
	DISPLAY DATA	The contents of the test section data array will be displayed for LENGTH bytes.
	DISPLAY SYS	The list of devices recognized by the MCP will be displayed.

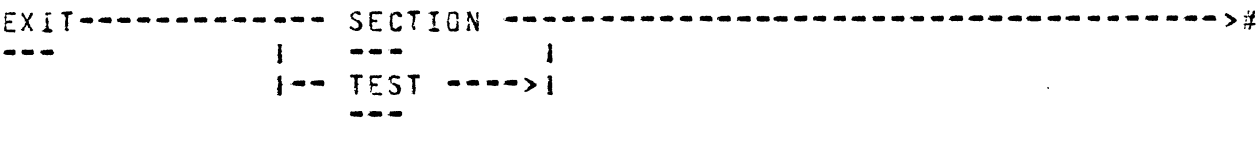


The ELOG command causes error messages to be stored on the system disk in the SYSTEM/ELOG file under "OPERATOR MESSAGES". Any result descriptors with exceptions will also be logged under the appropriate device. An error report may be obtained by performing an EI command to the MCP.

If ELOG is entered without the modifiers ON or OFF the option will be set on.

Default value: OFF

Examples:		
	ELOG	Error messages will be routed to the SYSTEM/ELOG file.
	ELO ON	Error messages will be routed to the SYSTEM/ELOG file.



The EXII command causes the section or list of sections (test) that are currently executing to be terminated. The values of options and variables will be unchanged.

If EXI SEC is entered, then the section currently executing will be terminated and control will pass to the next scheduled section.

If EXI TES is entered, then the current section, as well as all scheduled sections will be terminated. BOLT will go to the IDLE state and wait for operator instructions.

If a GO command is entered after an EXII TEST command has been performed, the next section to be executed will be the first entry in the section list.

This command has no effect if test sections are not currently being executed.

Example:	EXIT SEC	Terminate section currently executing and continue with next scheduled section.
	EXIT TEST	Terminate all scheduled sections and go to the IDLE state.

****** NOTE ******

Any input following this command on the same line will be ignored by BOLT.

GO----------->#
--

The GO command causes the initiation of the selected sequence of test sections, or the continuation of a test section sequence that was interrupted by operator intervention or a HALT command.

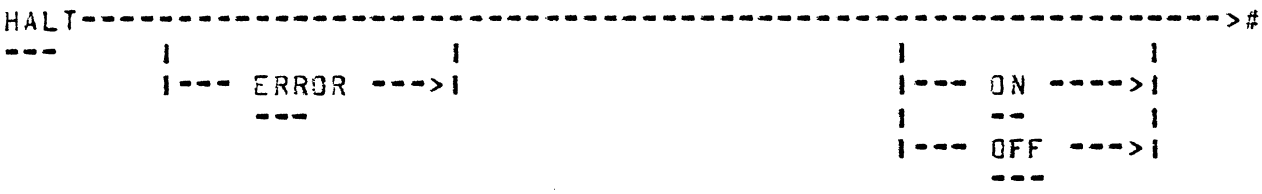
The UNIT and SECTION commands are used to build a list of test sections to be executed. The GO command starts the execution of the selected sections.

If a UNIT or SECTION command had been entered before the GO command, the execution will begin with the first section in the section list.

Example: GO

**** NOTE ****

Any input following this command on the same line will be ignored by BOLT.



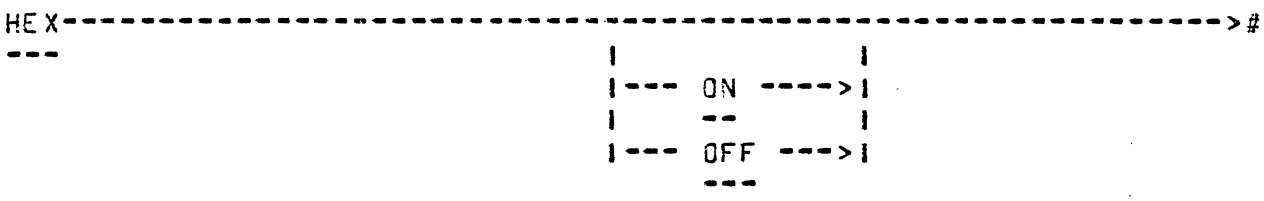
The HALT command causes the program to wait for a GO command after each error check routine and/or I/O descriptor list has completed.

The HALT ERROR command causes the program to wait for a GO command after each I/O descriptor list has completed, or error check routine if an error condition exists.

Both options can not be selected at the same time. If one is selected, the other is automatically turned off. If HALT or HALT ERROR are entered without the modifiers ON or OFF the options will be set on.

Default value: OFF

Examples:		
	HALT	BOLT will go to an idle state after each I/O descriptor and wait for operator input.
	HALT ERR ON	BOLT will go to an idle state after error and wait for operator input.
	HALT OFF	No longer halt after each I/O descriptor is issued.



The HEX command permits data to be displayed in hexadecimal format.

If this option is set when data is being displayed, the data will be displayed in hexadecimal format.

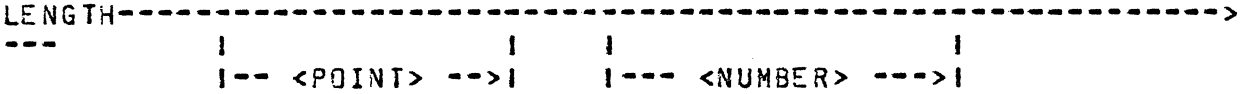
This command is independent of the ALPHA command, which can cause data to be displayed in alphabetic format.

It should be noted that this command does not display the data. It only controls the format of the data when it is displayed.

If HEX is entered without the modifiers ON or OFF the option will be set on.

Default value: ON

Examples:		
	HEX OFF	Data will not be displayed in hexadecimal format.
	HEX	Data will be displayed in hexadecimal format.



The LENGTH command is used to select the length of the data buffer to be read, written, displayed, or compared for PCAP descriptors or test sections.

If LENGTH is not followed by a number, the test program will use a default value for the data length. If LENGTH is followed by a value then those test sections that are "operator sensitive" to length will use the new value specified. Sections not sensitive to that parameter will not be affected.

The <POINT> is used to indicate that this command applies to a specific PCAP descriptor. If <POINT> is not used, then LENGTH refers to the data being used by the current test section.

The value of LENGTH for test sections can be displayed using the DISPLAY OPTIONS command. The value of LENGTH for a PCAP descriptor can be displayed using the DISPLAY DESC [#] command.

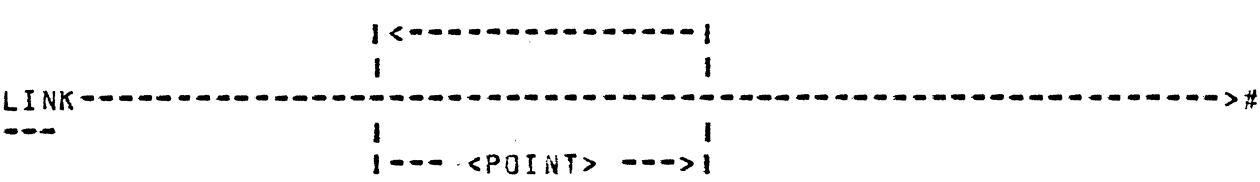
Default value: Device dependent

Minimum value: Device dependent

Maximum value: Device dependent

Examples:

LENGTH 100	The test section data array length will be 100 bytes.
LEN	A default value will be used as the length of the test section data array.
LEN [1] @30@	The data array length will be 48 bytes for PCAP descriptor [1].



The LINK command is used to link PCAP descriptors into a chain. The linking sequence is specified by a series of at least two <POINT> modifiers.

If LINK is entered without any modifiers, the link field of each PCAP I/O descriptor is set to null.

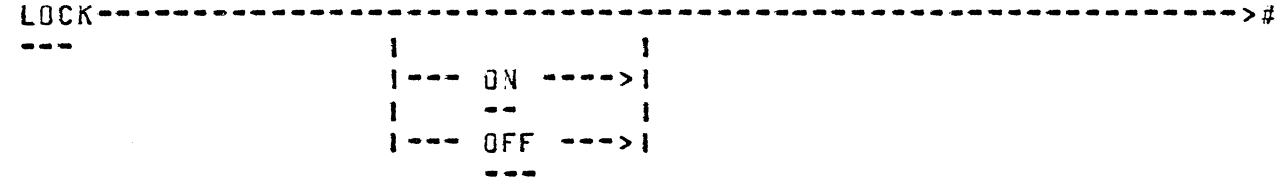
If LINK is followed by a single <POINT> modifier, only the link field in the designated PCAP descriptor will be set to null.

Each PCAP descriptor has only one link field, so a command such as LINK [2] [1] [2] [0] would be unclear. BOLT does not know if the link should be made from descriptor 2 to 1 or from 2 to 0. BOLT would store only the 2 to 0 link.

Circularly linked ops such as LINK [0] [1] [0] are not allowed. A check for circularly linked ops is done in the SEND command just before the ops are dispatched.

Examples:

LINK [2] [3]	Create a link from PCAP descriptor 2 to PCAP descriptor 3.
LINK [5]	Break the link from PCAP descriptor 5.
LINK	Break the links of all the PCAP descriptors.
LINK [3] [0] [2] [1]	Create the following links between the indicated PCAP descriptors: 3 to 0, 0 to 2, 2 to 1.



The LOCK command causes a loop to be set up from the beginning of a test section until the first error is detected. This loop will be repeated continuously until LOCK is reset, or the EXIT, CLEAR or RESTART command is entered.

Once an error is detected, it need not be present for the test section to remain in the loop. Additional errors that may occur after the LOCK loop has begun will not change the LOCK condition.

This command is useful in locating and scoping intermittent errors.

If LOCK is entered without the modifiers ON or OFF the option will be set on.

Default value: OFF

Examples:	LOCK	The sequence of operations which occurred before an error was detected will be continuously repeated.
	LOCK OFF	The test section will continue with the next succeeding operation even if an error is detected.

<LOGICAL OP>---------->#

Logical ops are two or three letter immediate action PCAP commands that cause a predefined op to be sent out to the selected unit. BOLT builds these ops in DESC [0]. DATA [0] and LENGTH [0] may be specified by the operator if required. The result of the Logical command may be displayed by entering DISP DESC [0]. The REPEAT and RETRY options apply to Logical PCAP commands in the same manner as physical PCAP commands.

BOLT verifies that the Logical command entered is valid for the device selected. Following are the lists of commands supported for each device:

TAPE

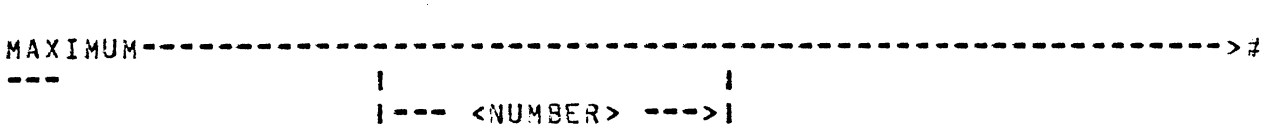
WRI - Write	RR - Read Reverse
REA - Read	SE - Space forward to EOF
REW - Rewind	TES - Test
UNL - Unload	WTM - Write Tape Mark
SPA - Space	TNR - Test Not Ready
BS - Backspace	TR - Test Ready
BSE - Backspace to EOF	ERA - Erase

PRINTER	CARD DEVICE	MULTILINE
TES - Test	TES - Test	TES - Test
WRI - Write	REA - Read	STO - Stop
FFD - Form Feed	WRI - Write	DSC - Disconnect
SKI - Skip one line		BRE - Break

For more information concerning the use of Logical commands, refer to the BOLT product specification for the device being tested.

****** NOTE ******

Any input following a Logical PCAP command on the same line will be ignored by BOLT.



The MAXIMUM command permits the operator to set a limit on the number of errors that may occur on a single unit before testing if forced to terminate on that unit.

If MAXIMUM is set to zero, the program will not check the number of errors.

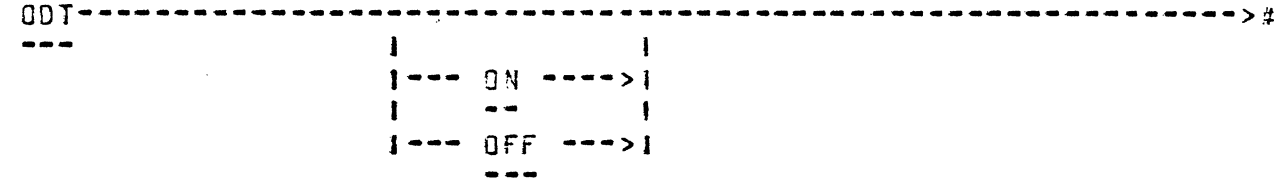
If MAXIMUM is entered without a number, it it assigned a value of zero.

Default value: 0

Minimum value: 0

Maximum value: 8388607

Examples:	MAXIMUM 0	There is no limit on the number of errors which a unit may receive.
	MAX 1	Once an error is detected during the testing of a unit, the testing of that unit is terminated.
	MAX 200	Once the number of errors detected on a single unit reaches 200, that unit will no longer be tested.



Both the ODT and PRINTER commands provide a means of selecting the destination of output messages from the test program.

If ODT is selected, output messages will be displayed upon the ODT.

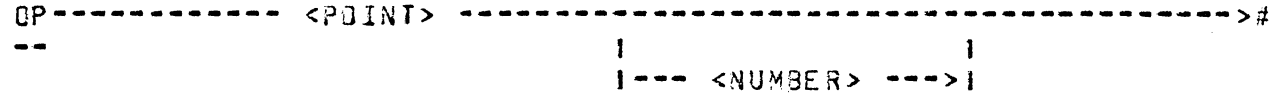
If both ODT and PRINTER are off, output messages will not be displayed. However, the test program will display ACCEPT messages on the ODT when it requires information to be entered by the operator.

In some cases messages displayed on the ODT will be condensed versions of those displayed on the line printer.

If ODT is entered without the modifiers ON or OFF the option will be set on.

Default value: ON

Examples:	ODT ON	Display output messages on the ODT.
	ODT OFF	Only messages necessary for BOLT operation will be displayed upon the ODT.



The OP command is used to select the operation code to be performed when building PCAP descriptors. The SEND command is used to dispatch descriptors built with the OP command.

If OP is not followed by a number, it will be assigned a value of zero.

The unit field in the op code does not need to be specified since BOLT automatically overwrites the last four bits of the op code with the unit number before dispatching the op.

The <POINT> modifier is used to indicate which PCAP descriptor this command applies to.

Default value: @0000000@

Minimum value: @0000000@

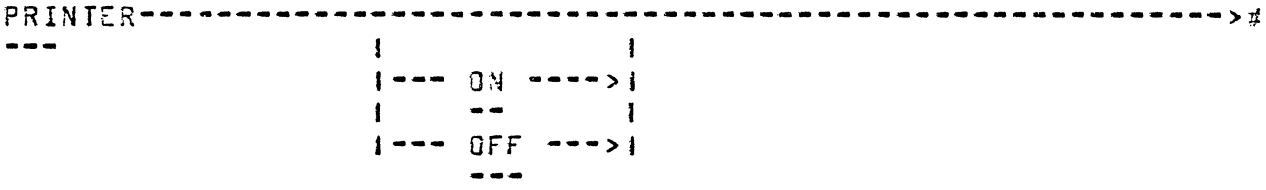
Maximum value: @FFFFFFF@

Examples:

OP [0]	Use operation code @0000000@ for PCAP descriptor [0].
OP [1] @8000000@	Use operation code @8000000@ for PCAP descriptor [1].
OP [5] 32	Use operation code @0000020@ for PCAP descriptor [5].

****** NOTE ******

The operator has the capability to enter invalid ops which may cause an undefined result on the device under test.



Both the ODT and PRINTER commands provide a means of selecting the destination of output messages from the test program.

If PRINTER is selected, output messages will be displayed upon the line printer. If a printer is not available, a BACKUP PRINT file will be created.

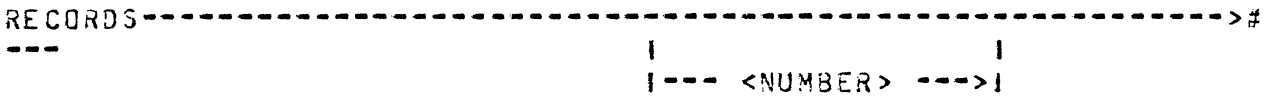
If both ODT and PRINTER are off, output messages will not be displayed. However, the test program will display ACCEPT messages on the ODT when it requires information to be entered by the operator. Selecting PRINTER OFF closes and releases an open printer file.

If PRINTER is entered without the modifiers ON or OFF the option will be set on.

Default value: OFF

Examples:

PRINTER ON	Display output messages on the line printer.
PRINTER OFF	The program will not display messages on the line printer. If a printer file is currently open, it will be closed and released.



The RECORDS command is used for certain test sections to select the number of physical records to be written or read on magnetic tape units or the number of lines to be written to a printer device.

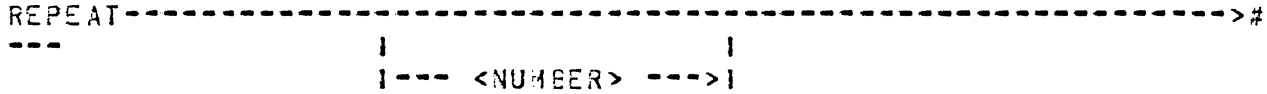
If RECORDS is not followed by a number, it will be assigned a value of one.

Default value: 1

Minimum value: 1

Maximum value: 8388607

Examples:		
	REC 10	The number of records is ten.
	RECORDS 2	The number of records is two.
	REC @40@	The number of records is 64.



Each section specified in the SECTION command, or each string of PCAP commands will be executed the number of times indicated by the REPEAT command.

For test sections a "BOJ" message will be displayed only at the beginning of a newly selected section. An "EOJ" message will be displayed only when the section has been repeated the selected number of times. Therefore, "BOJ" and "EOJ" will be displayed only once for each unique test section selection, since they are independent of the REPEAT value.

Whenever a SEND is entered for a linked list of physical PCAP descriptors, or whenever a logical PCAP op is entered, they will be issued the number of times specified in the REPEAT command.

If REPEAT is entered without a number, it is assigned a value of one, which means execute the specified operation one time.

(Also, see the CYCLE command)

Default value: 1

Minimum value: 1

Maximum value: 8388607

Example: REPEAT 1000 — Execute each section 1000 times when the GO command is entered, or each linked list of PCAP descriptors 1000 times when the next SEND command or Logical op is entered.

****** NOTE ******

Any input to BOLT while repeating PCAP operations will cause the repeating PCAP sequence to be terminated. Any input to BOLT while executing a repeating test section will cause the section to temporarily suspend until a GO is entered. The usual BOLT commands of RESTART, CLEAR or EXIT, however, will cause the repeating section to terminate.

RESTART---------->#

The RESTART command causes BOLT to be re-initialized. All options, values, data areas, unit lists and error totals will be reset, and restart messages will be displayed.

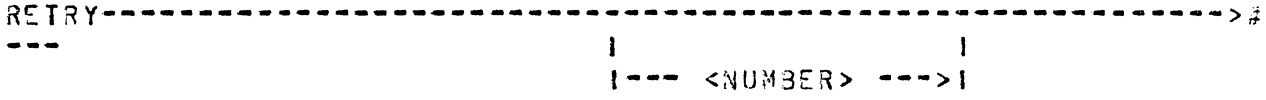
Any unit selected with the UNIT command will be released back to the control of the MCP.

Example: RESTART

RES

**** NOTE ****

Any input following this command on the same line will be ignored by BOLT.



The RETRY command is used to select the number of times that the test program will retry an I/O descriptor if a result status error is detected.

The program will retry the operation the number of times indicated by the RETRY command. Once the result error is not detected, the retries will be stopped.

In some test programs it is necessary to dispatch additional I/O descriptors immediately before the failing descriptor is retried. These descriptors are required to properly re-create the failure condition.

If error totals are recorded by the test program, retried I/O descriptors will not change these totals. A unit's error total will be incremented by one only if the failure still exists after the last retry.

If RETRY is entered without a number, it is assigned a default value. The default value for most devices is one. For the MAG TAPE, however, the default value is ten since under normal MCP conditions tape operations are retried this many times before errors are reported. When a different device is selected after a MAG TAPE, the RETRY value is reset to one, regardless of its value before the TAPE device was selected.

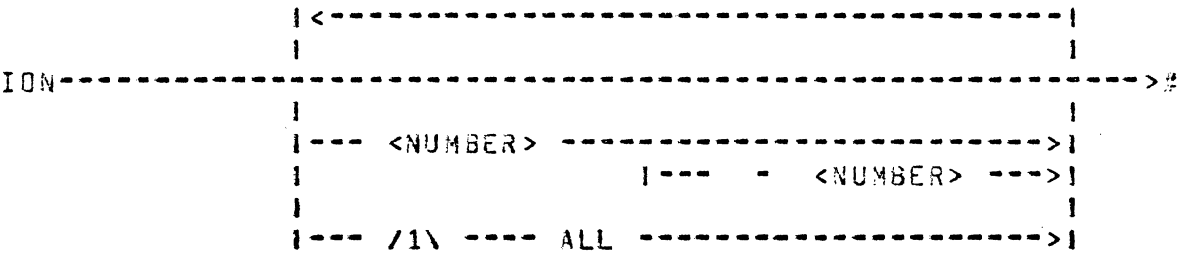
Default value: 1 (10 for MAG TAPE)

Minimum value: 1

Maximum value: 8388607

Examples:

RETRY	I/O descriptors will not be retried.
RET 10	I/O descriptors will be retried up to ten times.



The SECTION command builds a list of test sections to be executed.

Section names may be duplicated. The section designator on the right side of the dash must not be less than that on the left of the dash.

The command SECTION ALL causes all the default test sections to be selected.

Whenever a UNIT command is entered, the section list will be reset to its default value (default sequence of sections). The section list will be reset to its default value if SECTION is not followed by a number, number range, or the ALL modifier.

The default section list will contain confidence test sections only. Those sections that require operator interaction or perform diagnostic testing are optional and must be specifically selected by the operator.

A UNIT command must have been entered on the ODT prior to using the SECTION command. The optional diagnostic test sections require that the entire control be selected before testing, by using the BOLT UNIT CONTROL command.

Examples:	SECTION 2	Select section 2.
	SEC 1 - 9	Select Default Confidence Test sections 1,2,3,4,5,6,7,8 and 9.
	SEC 42 2 2 17	Select sections 42, 2, 2 and 17.
	SECTION ALL	Select a device dependent sequence of Default Confidence Test sections.

****** NOTE ******

The SECTION command may not be entered while a current section is in progress. The section must terminate normally or an EXIT command must be entered to bring the section to an orderly termination before a new section can be specified.

SEND---------- <POINT> ----------->#

The SEND command causes the sequence of physical PCAP descriptors built by the operator to be issued.

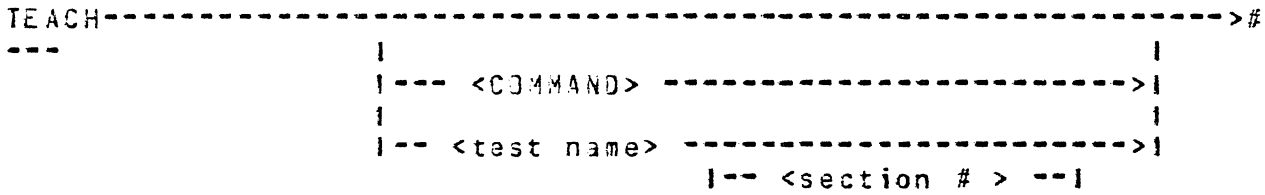
Values for OP, LENGTH, DATA and LINK should be specified before the SEND command is used.

The <POINT> modifier is used to indicate the first PCAP descriptor to be sent.

Example: SEND [3] Send the selected sequence of I/O descriptors, starting with descriptor number three.

****** NOTE ******

Any input following this command on the same line, will be ignored by BOLT.



Entering TEACH without any command specified, causes the test program to return a complete list of all valid commands and their valid abbreviations in the form:

<abbreviation> (<remaining letters>)

For example,

RESTART would appear as RES(TART)

Entering TEACH followed by a command will cause the test program to return a description of the syntax and semantics of the command.

Entering TEACH followed by a test name mnemonic will display brief instructions for the test and list all the valid section numbers and names. If the test name is followed by a valid section number, the program will display a description of the test performed by that section.

The TEACH command requires the data file BOLT/TEACH. BOLT may be run without the BOLT/TEACH file, but the operator will be notified that the file is missing when BOLT is started or when the TEACH command is entered.

Examples:	TEACH SECTION	A syntax diagram for the SECTION command is displayed along with some examples.
	TEACH MTA	A brief description of the MAG TAPE test is displayed specifying the valid section numbers and controls.
	TEA MT 3	A brief description of section 3 of the MAG TAPE test is displayed which includes operator sensitive variables.

The IRACE command causes selected information to be displayed during the execution of every PCAP and test section I/O descriptor.

If IRACE is selected without any modifiers then the command defaults to DESCRIP

If BUFFERS is selected, the data buffers sent or received from the control will be displayed before and after each I/O.

If DESCRIP is selected, the I/O descriptors sent to the control will be displayed before and after each I/O. The result status field will contain the result after the operation was completed.

If AUDIT is selected, the program DC/AUDIT will be fired up by BOLT and the internal BOLT file "AUDIT" will be opened and initialized. Every subsequent datacomm I/O will be logged in the AUDIT file and may be analyzed by the DC/AUDIT program. Diagnostic datacomm ops will not be traced by DC/AUDIT because they specify values in the A and B fields when DC/AUDIT expects buffer addresses. When DC/AUDIT is started by BOLT it will ask the operator for parameters. It is usually sufficient to enter IO.DESC=1 and then END. If other options are desired, then refer to the DC/AUDIT program documentation. Turning IRACE AUDIT OFF will inhibit the recording of datacomm I/O in the AUDIT file. The AUDIT file will be closed and DC/AUDIT brought to EOJ when BOLT goes to EOJ. This option is only used for DATACOMM tests.

Each command is independent of the others, and may be turned ON and OFF without affecting the other IRACE commands. However, IRACE OFF will turn all of the IRACE options off.

If the command is entered without the modifiers ON or OFF the option will be set on.

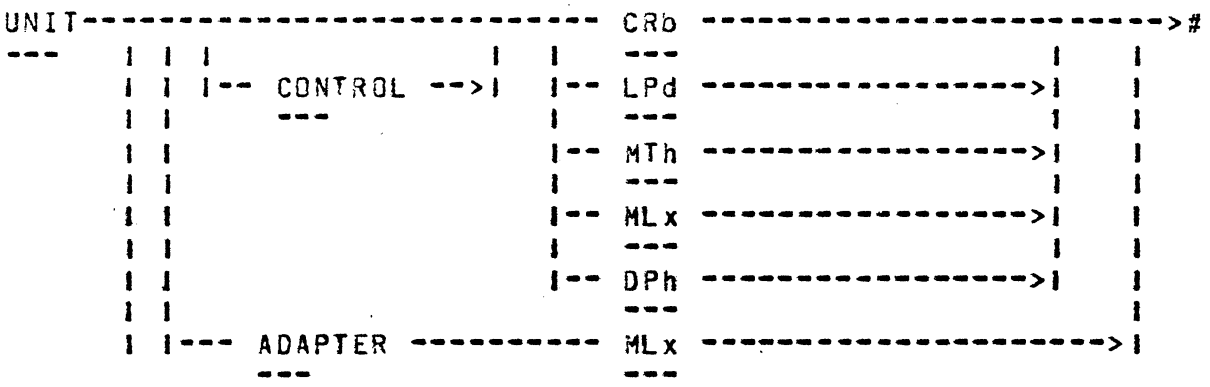
Default value: OFF

Examples:		
	TRACE	The I/O descriptors sent to the control will be displayed.
	TRACE BUF	The data buffers transferred to or from the control will be traced.

****** NOTE ******

Occasionally, when tracing a large number of operations (for example when REPEAT is set), the system may not be able to keep up with the display of information from BOLT and some trace information may be lost. One suggestion for a long sequence of operations is to set the HALT ootion and enter GO after each I/O to guarantee that the system has enough time to display all the information.

The TRACE options may cause time critical tests to report errors. These errors should always be verified by turning the TRACE options OFF and rerunning the test.



Where b = A or B.
d = A, B, C, D.
h = A, B, ..., H.
x = A - Z, 1 - 6.

The UNIT command selects the I/O device to be tested. If a mnemonic is used to designate the device, it must be valid. Where applicable it must also be MCP compatible.

The UNIT command resets the SECTION list to its default value.

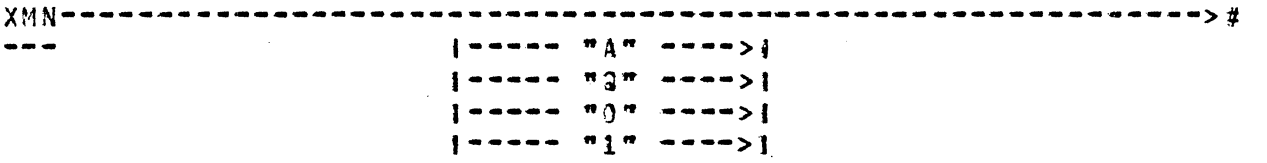
The CONTROL modifier is used to distinguish between the selection of a single unit on a control and the selection of the entire control. For MLx the CONTROL modifier selects Base card 0 if x=A and Base card 2 if x=Q. For system security reasons the entire peripheral control must be obtained before any PCAP commands may be issued to a unit on that control, or before a test section may execute which performs diagnostic ops on the control itself. BOLT verifies that a control has been selected before issuing a PCAP SEND command. The test routines verify that a control has been selected before executing any sections that perform control diagnostics.

The ADAPTER modifier applies only to MLx and is ignored by all other devices. ADAPTER opens four units beginning with the unit specified by "x".

If a valid unit is selected then BOLT will attempt to gain access to that device through a diagnostic OPEN. However, before that OPEN, a procedure is executed to return the currently selected device to its normal operating condition, and a diagnostic CLOSE is attempted by BOLT on the device in use. If the CLOSE fails then an error message is displayed telling why and the unit command will have no effect. If the CLOSE is successful but the subsequent OPEN (on the new device) is not, then an error message is displayed telling the operator that the unit was not secured for use by BOLT.

A SECTION command must be preceded by a UNIT command.

Examples:	UNIT LPA	Select line orinter LPA.
	UNIT CON MTB	Select the control that tape drive B is connected to. All drives connected to the same control will also be reserved, but all unit ops will only go to MTB.
	UNIT ADAPTER MLE	Select the second multiline adapter card connected to base card 0. All four units on that adapter will be opened.



MULTILINE DEVICES ONLY

The XMN command allows the operator to specify a terminal transmission number for datacomm test sections that may optionally use them.

The only valid parameters for the XMN command are "A", "3", "0" or "1". When any of these are entered, the USE_XMISSION_NUM flag is set and alternating pairs of transmission numbers (A and 3, or 0 and 1) are sent to the terminal during write operations. The terminals will ignore the numbers except for the calculation of the message BCC.

When XMN is entered with no parameter, the USE_XMISSION_NUM flag is reset and the transmission numbers are not used.

Example:	XMN "1"	Transmission numbers of alternating 1's and 0's are sent to the terminal by those confidence test sections that may use them.

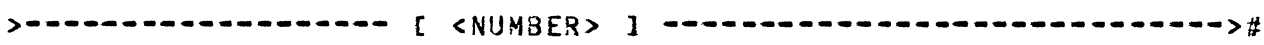
A sample select message would be:

EOT AD1 AD2 FSL SOH AD1 AD2 3F3 STX...ETX

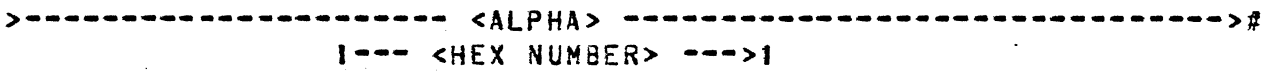
XMN	No transmission number is used.

LITERALS AND VALUES

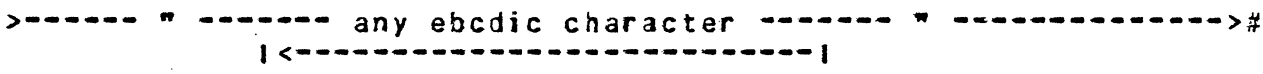
POINT



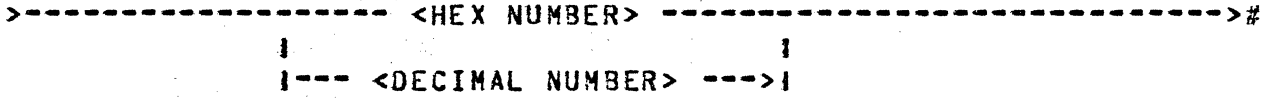
DATA GROUP



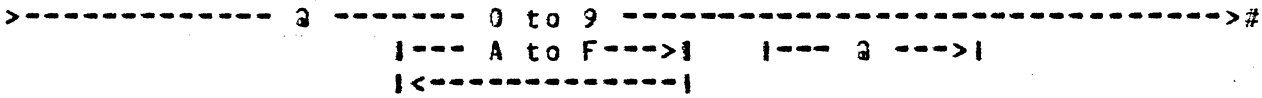
ALPHA



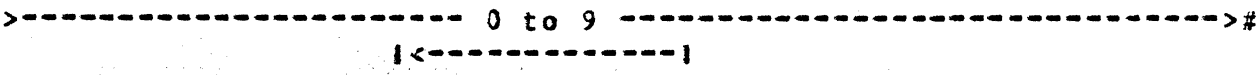
NUMBER



HEX NUMBER



DECIMAL NUMBER



BURROUGHS CORPORATION
COMPUTER SYSTEMS GROUP
SANTA BARBARA PLANT

OPERATING INSTRUCTIONS

ENTERING COMMANDS

BOLT is executed like any other normal state program under control of the MCP. When BOLT comes up it will display the following message on the ODT:

***** ENTER NEXT COMMAND *****

The operator may then communicate with BOLT by using the ACCEPT mechanism as follows:

<BOLT job number>AX <BOLT command>

Whenever BOLT goes to the idle state or wants operator input it will display the ENTER NEXT COMMAND message on the ODT. The operator may also enter unsolicited input at any time in order to stop or change a test.

When BOLT is executed, or the CLEAR or RESTART commands are entered, the display and operating options are initialized to their default values. The operator may change the options at any time and the current state of the program options may be queried using the DISPLAY command.

REQUIRED FILES

BOLT requires a minimum of two files to be present in order to be executed: The SDL object file named "BOLT", and a random disk input data file named "BOLT/ERROR". In addition the optional random disk input data file named "BOLT/TEACH" should be present if the TEACH command is to be used. If "BOLT/TEACH" is not present, BOLT may still be executed but the operator will be informed that the file is missing when BOLT is started and every time the TEACH command is entered.

PREDEFINED TESTS

To execute a predefined confidence test on an I/O device, the operator would first select the desired device using the UNIT command. If the device is not available then the operator must wait for the device to become available or test another device.

Once the device has been properly selected using the UNIT command, the operator may execute the default list of test sections, using the default setting of the options by simply entering the GO command. The test will report the number of errors found for each section and display a message when the test has finished.

If any options other than the defaults are desired, then the operator would enter the option command(s) and then enter the GO command. The default section list for each test will consist of confidence test sections that do not require operator intervention. For instructions on executing optional test sections that perform diagnostic testing or further confidence testing which requires operator intervention, refer to the SECTION command documentation.

EXAMPLE TEST RUN

This section shows the BOLT commands used to run a typical test session on the LINE PRINTER. Also shown are BOLT's output to the ODT in response to these commands.

First BOLT is executed, at which time it displays some initialization messages. When BOLT is ready for input it will prompt the operator for an AX message:

```
EX BOLT
: BOLT = 325 BOJ. PP=4, MP=4 TIME = 14:55:04.2
: % BOLT =325 BOLT (B1000 ON-LINE TESTS) VERSION: 1.1  DATED: 01/23/81
: % BOLT =325 ENTER "<MX #>AX TEACH" FOR A SUMMARY OF BOLT COMMANDS.
: % BOLT =325 *****  ENTER NEXT COMMAND  *****
: BOLT =325 ACCEPT.
```

Next a device is selected using the UNIT command which executes initialization code for that device and displays some greeting messages if the unit selected was successfully obtained:

```
325AX UNIT LPA
: % BOLT =325
: % BOLT =325
: % BOLT =325
: % BOLT =325 TOTAL UNITS OBTAINED = 01
: % BOLT =325 UNIT NAME = LPA    PORT = 7 CHANNEL = 3 UNIT = 0
: % BOLT =325
: % BOLT =325              LINE PRINTER TEST -- VERSION AA
: % BOLT =325              ------------------------------
: % BOLT =325
: % BOLT =325       DEFAULT  CONFIDENCE TEST SECTIONS ARE   1 - 8
: % BOLT =325       OPTIONAL CONFIDENCE TEST SECTIONS ARE  25 - 32
: % BOLT =325       OPTIONAL DIAGNOSTIC TEST SECTIONS ARE  50 - 62
: % BOLT =325          (50 - 53 FOR PC 5/6,  54 - 62 FOR PC 7)
: % BOLT =325
: % BOLT =325 FOR A DESCRIPTION OF THE LINE PRINTER TEST ENTER "TEACH
: % BOLT =325 LP".  ENTER "TEACH LP <SECTION-NUMBER>" FOR THE
: % BOLT =325 DESCRIPTION OF ANY TEST SECTION LISTED ABOVE.  SECTIONS
: % BOLT =325 1-8 ARE READY TO RUN.
: % BOLT =325 *****  ENTER NEXT COMMAND  *****
: BOLT =325 ACCEPT.
```

Once the device has been selected, the default confidence sections can be executed by simply entering the GO command. A beginning of test message is displayed for the unit and BOJ - EOJ messages are displayed for each section which includes the device name, section number and section name.

If an error is detected by a test section, the EOJ message is followed by the number of errors detected for that section. When all scheduled sections have completed (either normally or due to a terminating command), an EOJ message is displayed for the UNIT along with the total number or errors detected for all sections run. BOLT then prompts the operator for the next command:

```
325AX GO
: % BOLT =325 BEGINNING OF TEST FOR UNIT LPA
: % BOLT =325 LPA: BEGIN SECTION  1: ALPHANUMERIC RIPPLE TEST.
: % BOLT =325 LPA: END   SECTION  1: ALPHANUMERIC RIPPLE TEST.
: % BOLT =325 LPA: BEGIN SECTION  2: END ADDRESS TEST.
: % BOLT =325 LPA: END   SECTION  2: END ADDRESS TEST.
: % BOLT =325 LPA: BEGIN SECTION  3: BROADSIDE TEST.
: % BOLT =325 LPA: END   SECTION  3: BROADSIDE TEST.
: % BOLT =325 LPA: BEGIN SECTION  4: NO PAPER ADVANCE TEST.
: % BOLT =325 LPA: END   SECTION  4: NO PAPER ADVANCE TEST.
: % BOLT =325 LPA: BEGIN SECTION  5: VERIFY CHANNELS TEST.
: % BOLT =325 LPA: END   SECTION  5: VERIFY CHANNELS TEST.
: % BOLT =325 LPA: BEGIN SECTION  6: FASTEST PATTERN TEST.
: % BOLT =325 LPA: END   SECTION  6: FASTEST PATTERN TEST.
: % BOLT =325 LPA: BEGIN SECTION  7: SINGLE SPACE WRITE TEST.
: % BOLT =325 LPA: END   SECTION  7: SINGLE SPACE WRITE TEST.
: % BOLT =325 LPA: BEGIN SECTION  8: DOUBLE SPACE WRITE TEST.
: % BOLT =325 LPA: END   SECTION  8: DOUBLE SPACE WRITE TEST.
: % BOLT =325 END OF TEST ON UNIT LPA -- 000000 ERRORS REPORTED.
: % BOLT =325 *****  ENTER NEXT COMMAND  *****
: BOLT =325 ACCEPT.
```

Optional Confidence test sections can be executed by selecting the section number or range and entering GO. These sections usually require some type of operator interaction such as making the device READY or visually verifying the result:

```
325AX SECTION 25; GO
: % BOLT =325 BEGINNING OF TEST FOR UNIT LPA
: % BOLT =325 LPA: BEGIN SECTION 25: CHARACTER ALIGNMENT TEST.
: % BOLT =325 PLEASE READY THE LINE PRINTER BEING TESTED.
: % BOLT =325 ENTER "GO" AFTER THE LINE PRINTER HAS BEEN MADE READY.
: % BOLT =325 *****  ENTER NEXT COMMAND  *****
: BOLT =325 ACCEPT.
```

At this point the operator would make the PRINTER READY and enter GO:

```
325AX GO
: % BOLT =325 CHARACTER ALIGNMENT PATTERN HAS BEEN WRITTEN TO PRINTER.
: % BOLT =325 PLEASE VERIFY THAT CHARACTER ALIGNMENT IS CORRECT.
: % BOLT =325 LPA: END   SECTION 25: CHARACTER ALIGNMENT TEST.
: % BOLT =325 END OF TEST ON UNIT LPA -- 000000 ERRORS REPORTED.
: % BOLT =325 *****  ENTER NEXT COMMAND *****
: BOLT =325 ACCEPT.
```

When an error occurs, BOLT informs the operator by displaying the device, section # and error # along with a brief message. Further error information can usually be found in the BOLT product specification for that device. The op code issued, result status and interpretation of the status bits are also displayed. Whenever possible, at this point the section will either terminate or request further information from the operator:

```
325AX SECTION 2;GO
: % BOLT =325 BEGINNING OF TEST FOR UNIT LPA
: % BOLT =325 LPA: BEGIN SECTION  2: END ADDRESS TEST.
: % BOLT =325  LPA:SEC 02:ERR# 02: - THE SINGLE SPACE WRITE OP FAILED.
: % BOLT =325 THE OP CODE ISSUED WAS: 4E0000
: % BOLT =325 IO RESULT STATUS FIELD: E00080
: % BOLT =325 THE FOLLOWING STATUS BITS ARE ON:
: % BOLT =325    RS BIT      1:  OPERATION COMPLETE
: % BOLT =325    RS BIT      2:  EXCEPTION
: % BOLT =325    RS BIT      3:  DEVICE NOT READY
: % BOLT =325    RS BIT     17:  OPERATION COMPLETE
: % BOLT =325 THE DEVICE WENT NOT READY DURING A TEST WHICH NEEDS IT
: % BOLT =325 READY.
: % BOLT =325 PLEASE MAKE THE LINE PRINTER READY. THEN ENTER THE "GO"
: % BOLT =325 COMMAND.
: % BOLT =325 *****  ENTER NEXT COMMAND  *****
: BOLT =325 ACCEPT.
```

When testing is completed for one device another device can be selected using the UNIT command or BOLT can be brought to end of job by using the BYE command. Before going to EOJ, BOLT will release the device under test along with any backup files that have been created:

```
362AX BYE
: % BOLT =325 RESTORING PRINTER DEVICE BEFORE IT IS RELEASED
: % BOLT =325 LPA RELEASED BY BOLT.
: % BOLT =325 EOJ. TIME = 15:30:47.7
```

PCAP TESTING USING LOGICAL I/O COMMANDS

In addition to running predefined tests, BOLT gives the operator the ability to build and issue individual descriptors or chains of descriptors, through the PCAP facility. All PCAP commands require that the device to be tested, along with its associated control, be obtained by BOLT using the UNIT CONTROL command.

The simplest level of PCAP operations are called LOGICAL commands which when entered cause a predefined op to be built in DESC [0] and causes the descriptor to be immediately issued. (Refer to the LOGICAL OP command description.)

For example, a tape device can be rewound with the following command:

```
76AXREWIND
 % BOLT =76 *****  ENTER NEXT COMMAND *****
 BOLT =76 ACCEPT.
```

Or a printer can be made to skip to the top of a page with a Form Feed command:

```
76AXFFD
 % BOLT =76 ***** ENTER NEXT COMMAND *****
 BOLT =76 ACCEPT.
```

LOGICAL commands can also be used to read and write to devices. The WRITE command requires that DATA [0] and LENGTH [0] first be specified by the operator:

```
76AXDATA [0] "THIS IS A LOGICAL PCAP EXAMPLE FOR MAG TAPE"
 % BOLT =76 *****  ENTER NEXT COMMAND  *****
 BOLT =76 ACCEPT.
76AXLENGTH [0] 100
 % BOLT =76 *****  ENTER NEXT COMMAND  *****
 BOLT =76 ACCEPT.
76AXWRITE
 % BOLT =76 *****  ENTER NEXT COMMAND  *****
 BOLT =76 ACCEPT.
```

Since LOGICAL commands use DESC [0] the DISPLAY command can be used to look at the RESULT of the operation:

```
:
76AXDISP DESC [0]
 % BOLT =76                          DESCRIPTOR 0
 % BOLT =76                          ------------
 % BOLT =76
 % BOLT =76  OP:      400001   A_FIELD:  009E7F   RESULT:     800080
 % BOLT =76  LINK: -0000001   B_FIELD:  00A19F   XRSLT1:   00000000
 % BOLT =76  PRT/CH:     7C   C_FIELD:  000000   XRSLT2:   00000000
 % BOLT =76                   LENGTH:   000064 (000100)
 % BOLT =76
 % BOLT =76 *****  ENTER NEXT COMMAND  *****
, BOLT =76 ACCEPT.
```

LOGICAL commands are quick and easy to use since they do not require the operator to know the bit patterns for commonly used ops. However, LOGICAL ops are limited in that they may not be linked together and the variant bits may not be specified by the operator. If more flexibility is desired, the operator should use the PHYSICAL PCAP commands described below. Keep in mind that LOGICAL PCAP commands will always overwrite any PHYSICAL PCAP information that has been stored in DESC [0].

PCAP TESTING USING PHYSICAL I/O COMMANDS

The operator may also perform a custom test on the selected peripheral device by using the PHYSICAL PCAP facility of BOLT. The operator must first select the device and its entire control by using the CONTROL modifier of the UNIT command. Once the device and control have been reserved, the operator builds I/O descriptors using PHYSICAL PCAP commands such as OP, DATA, and LINK and then dispatches them using the SEND command. The operator may build from 1 to 6 descriptors, each with its own data buffer. The descriptors may be built, linked, and dispatched in any order. The current value of a descriptor field and its data buffer may be viewed at any time using the DISPLAY DESC and DISPLAY DATA commands. It is with these commands that the operator views descriptors which are being constructed and those which have been issued. The list of linked PCAP descriptors may be continuously sent out by using the REPEAT command.

For example, assume that the operator has selected a tape device using the command UNIT CONTROL MTA, and wishes to write a record, read it back and compare the data. First the necessary write, backspace and read op would be built in PCAP descriptor 1 thru 3 and linked together:

```
76AXOP [1] @4000000;OP [2] @D800000;OP [3] @0000000;
 % BOLT =76 *****  ENTER NEXT COMMAND  *****
 BOLT =76 ACCEPT.
76AXLINK [1] [2] [3];
 % BOLT =76 *****  ENTER NEXT COMMAND  *****
 BOLT =76 ACCEPT.
76AXDATA [1] "THIS IS A PCAP EXAMPLE "
 % BOLT =76 *****  ENTER NEXT COMMAND  *****
 BOLT =76 ACCEPT.
76AXLENGTH [1] 44;LENGTH [3] 44
 % BOLT =76 *****  ENTER NEXT COMMAND *****
 BOLT =76 ACCEPT.
76AXDATA [3] " "
 % BOLT =76 *****  ENTER NEXT COMMAND  *****
```

The descriptors and the data may then be verified using the display command:

```
76AXDISPLAY DESC [1]
 % BOLT =76                          DESCRIPTOR 1
 % BOLT =76                          ------------
 % BOLT =76
 % BOLT =76  OP:      400001   A_FIELD:  008E7F   RESULT:     000000
 % BOLT =76  LINK:    000002   B_FIELD:  008FDF   XRSLT1:   C0000000
 % BOLT =76  PRT/CH:     7C   C_FIELD:  000000   XRSLT2:   00000000
 % BOLT =76                   LENGTH:   00002C (000044)
 % BOLT =76
 % BOLT =76 *****  ENTER NEXT COMMAND  *****
76AXDISP DATA [1]
 % BOLT =76 DATA [1] CONTAINS THE FOLLOWING DATA
 % BOLT =76
 % BOLT =76 E3C8C9E240C9E240C140D7C3C1D740C5E7C1D4D7D3C540E3C8C9E2
 % BOLT =76 40C9E240C140D7C3C1D740C5E7C1D4D7D3
 % BOLT =76
 % BOLT =76 THIS IS A PCAP EXAMPLE THIS IS A PCAP EXAMPLE
```

Notice that BOLT provides the unit number in the last four bits of the op field. The result status field is cleared before any of the descriptors are issued.

The first descriptor in the chain is dispatched with the SEND command and the read and write buffers are checked against each other with the COMPARE command:

```
76AXSEND [1]
 % BOLT =76 *****  ENTER NEXT COMMAND  *****
 BOLT =76 ACCEPT.
76AXCOMPARE [1] [3]
 % BOLT =76 COMPARE OF BUFFER 1 TO BUFFER 3 WAS SUCCESSFUL
 % BOLT =76 *****  ENTER NEXT COMMAND  *****
```

Each of the descriptors can be displayed and the RESULT field examined to verify that the ops completed as expected:

```
76AXDISP DESC [1]
 % BOLT =76                          DESCRIPTOR 1
 % BOLT =76                          ------------
 % BOLT =76
 % BOLT =76  OP:      400001   A_FIELD:  008E7F   RESULT:     800080
 % BOLT =76  LINK:    000002   B_FIELD   008FDF   XRSLT1:   00000000
 % BOLT =76  PRT/CH:     70   C_FIELD   000000   XRSLT2:   00000000
 % BOLT =76                   LENGTH:   00002C (000044)
 % BOLT =76
 % BOLT =76 *****  ENTER NEXT COMMAND  *****
```

To demonstrate what would happen if a data miscompare was encountered on the read operation, one letter in DATA [3] will be changed and the two buffers will be compared again:

```
76AXDATA [3] "THIS IS A PHAP EXAMPLE"
 %BOLT =76 *****  ENTER NEXT COMMAND  *****
 BOLT =76 ACCEPT.
76AXCOMPARE [1] [3]
 % BOLT =76 MISCOMPARE BETWEEN EXPECTED AND OBSERVED DATA.
 % BOLT =76 DATA COMPARISON ERROR - FIRST ERROR DETECTED AT
 % BOLT =76 HEX 00023
 % BOLT =76 MAX OF 46 CHARACTERS ARE DISPLAYED STARTING AT
 % BOLT =76 THE 1ST DETECTED ERROR.
: % BOLT =76
 % BOLT =76 EXPECTED(00011):HAP EXAMPLE THIS IS A PHAP EXAMPL
 % BOLT =76 OBSERVED        :CAP EXAMPLE THIS IS A PCAP EXAMPL
 % BOLT =76 ERROR(S)        :*                          *
 % BOLT =76
 % BOLT =76
 % BOLT =76 EXPECTED(00022):C8C1D7 40C5E7 C1D4D7 D3C540 E3C8C9 E240C9
 % BOLT =76 OBSERVED        :C3C1D7 40C5E7 C1D4D7 D3C540 E3C8C9 E240C9
 % BOLT =76 ERROR(S)        : *
 % BOLT =76
 % BOLT =76 EXPECTED(00058):E240C1 40D7C8 C1D740 C5E7C1 D4D7D3 C5
 % BOLT =76 OBSERVED        :E240C1 40D7C3 C1D740 C5E7C1 D4D7D3 C5
 % BOLT =76 ERROR(S)        :            *
 % BOLT =76
 % BOLT =76 TOTAL NUMBER OF 4-BITS IN ERROR, 00002
 % BOLT =76 COMPARE OF BUFFER 1 TO BUFFER 3 WAS NOT SUCCESSFUL
 % BOLT =76 *****  ENTER NEXT COMMAND  *****
```

CONTROLS AND DEVICE TYPES

To better understand the steps involved in running an On-Line test routine, consider the following example. Assume that an operator, running BOLT from an ODT, wishes to run the Default Confidence Confidence Test Sections on two different devices (MTA and LPA for example). After the completion of both tests the operator wants to bring BOLT to END-OF-JOB (EOJ). See FIGURE 1 below.

OPERATOR INPUT	BOLT RESPONSE
UNIT MTA	Verify that the specified unit is valid. Perform a diagnostic OPEN on the MTA unit. If successful OPEN then access is granted, and Default Section numbers are loaded. If unsuccessful OPEN then an error message is displayed and UNIT MTA has no effect.
GO	Call the TEST_UNIT procedure. Do all test sections specified and return control. TEST_UNIT calls MAGNETIC_TAPE_TEST with the section number to be executed.
	***** Magnetic Tape Test Finished *****
UNIT LPA	Verify that the specified unit is valid. Perform a diagnostic CLOSE on unit MTA, which releases it back to MCP control. If successful CLOSE then attempt OPEN LPA. If unsuccessful CLOSE then an error message is displayed and UNIT LPA has no effect. Perform a diagnostic OPEN on the LPA unit. If successful OPEN then access is granted, and Default Section numbers are loaded. If unsuccessful OPEN then an error message is displayed and UNIT LPA has no effect.
GO	Call the TEST_UNIT procedure. Do all test sections specified and return control. TEST_UNIT calls LINE_PRINTER_TEST with the section number to be executed.
	******* Line Printer Test Finished *******
BYE	Perform a diagnostic CLOSE on LPA unit. If successful CLOSE then bring BOLT to EOJ. If unsuccessful CLOSE then an error message is displayed and the BYE has no effect

FIGURE 1 - GAINING ACCESS TO DEVICES AND RUNNING TESTS. This figure shows the steps involved in gaining access to a device when the UNIT command is entered. Also shown is the minimum operator input required to run Default Confidence Test Sections for a device which has them.

PROGRAM SWITCHES

For debugging and tracing purposes the software program switches for BOLT have been assigned as follows:

SW 0 - BOLT Executive (reserved)
SW 1 - BOLT Executive (TTT macro)
SW 2 - BOLT Executive (DDD macro)
SW 3 - BOLT Executive (reserved)
SW 4 - BOLT Executive (reserved)

Likewise, program switches five (5) through nine (9) have been assigned to the On-Line Test Routines as follows:

SW 5 - Trace Procedure Entries and Exits
SW 6 - Debug Output Such As:
-- Procedure Parameters
-- Important Variables
-- Procedure and Function Results
SW 7 - On-Line Test Routine (reserved)
SW 8 - On-Line Test Routine (reserved)
SW 9 - On-Line Test Routine (reserved)

All other program switches are either used internally or are not used by the BOLT Executive and/or On-Line Test Routines. In either case, those program switches should never be enabled when executing BOLT.

DYNAMIC MEMORY ALLOCATION

The default allocated dynamic core for BOLT is 1024 bytes (8192 bits). This area is used as a buffer for Magnetic Tape operations (Read, Write & Erase); however, the operator can increase the size of the buffer using the MCP MO command.

Example: MO BOLT ME 16384 Sets the buffer size to 2048 bytes.

The maximum dynamic memory that can be allocated for an SDL program such as BOLT is 4K-1 bytes (4095 bytes or 32760 bits). When BOLT is executed it will check its dynamic memory, display an error message and go to End of Job if it is less than 1024 bytes or greater than 4095 bytes. It should be mentioned that the LENGTH command can be used to adjust the length of the buffer.

ODT TRACE

BOLT's ODT output messages are designed to work with ODTs or remote terminals with ODT capabilities. The messages are designed to scroll from the bottom of the screen to the too. Since the default mode of the ODT is to scroll from top to bottom, the BOLT messages will appear in reverse order on the screen. This can be corrected by entering the MCP command KB DIRN 1 which alters the ODT default mode and causes the messages to scroll from bottom to top. To return the ODT to default mode, enter KB DIRN 0.

Sometimes it is desirable to have a permanent record of the commands entered to BOLT through the ODT as well as BOLT's responses to the ODT. This can be done by making the printer NOT READY and entering the MCP command KB LP ON. This will create a printer backup file called "SYSTEM.ODT/OUTPUT" which will be a trace of all ODT activity. When all the desired information has been recorded the operator may enter KB LP OFF and PB SYSTEM.ODT/OUTPUT.

PROGRAM STRUCTURE

The BOLT executive and test procedures are compiled as a single SDL program. Each test is written as one large procedure that is inserted into the executive. The entire executive, including the test procedures, is BOLT/SOURCE. The organization of this file is shown in FIGURE 2.

When BOLT/SOURCE is compiled, it generates the listing file BOLT/LIST and the object file BOLT/OBJ. In order to execute BOLT/OBJ, the two data files BOLT/TEACH and BOLT/ERROR are required. These files contain the teach and error messages, respectively, for BOLT.

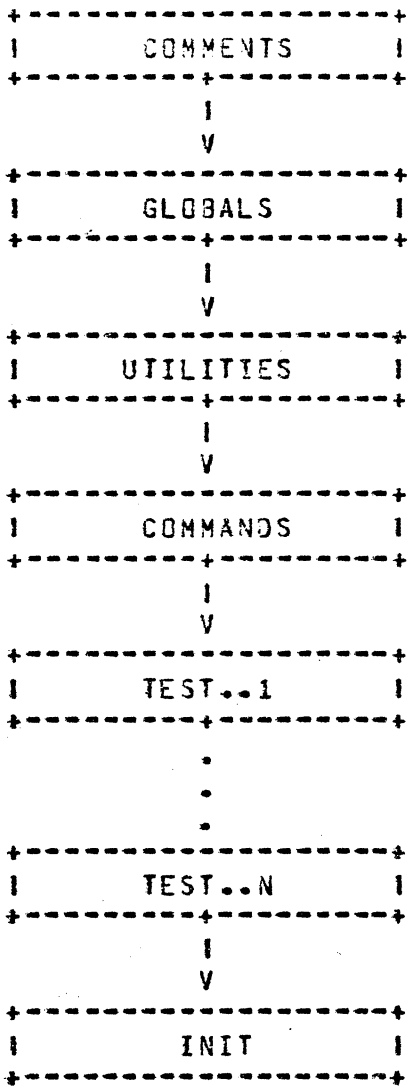


FIGURE 2 - BOLT STRUCTURE. The source code for BOLT is arranged in the order shown above. The test procedure for each device is inserted between the command procedures and the intialization code.

COMMENTS

The COMMENTS section contains all documentation that appears at the front of the program such as copyright and release information.

GLOBALS

The GLOBALS section contains the global defines and declares for BOLT. Defined as constants are such values as array lengths, valid unit mnemonics, diagnostic I/O constants, and the limit and default values for parameter options. Error message names are also declared as sequential numbers for indexing into the error file. The error display and diagnostic I/O routines that are called by the test sections are also located in this file.

UTILITIES

The UTILITIES section contains utility procedures for the executive such as SCAN, FILL_BUF, COMPARE_BUFF, GET_INDEX, etc. Procedures that are common among test routines are also found here.

COMMANDS

The COMMANDS section contains the procedures that are called by INPUT_CONTROL to process a command that has been entered by the operator at the ODT.

TESTS

Each type of I/O device will have a separate test that is written as a BOLT procedure. The procedure consists of a case statement which does a call to a sub-procedure for each section of the test. The executive determines the section number to execute and passes the number to the test procedure. When the selected test section has complete, control is returned to the executive. The test procedure sections interface to the executive by using global procedures and global variables that are maintained by the executive.

INITIALIZE

The INIT section contains the MAIN procedure, the INPUT_CONTROL procedure, and the TEST_UNIT procedure. The MAIN code initializes arrays and variables and sets up in an infinite loop that sets the current section index to zero and calls INPUT CONTROL. INPUT CONTROL displays a message to the operator asking for the next command, and then waits for the operator to enter input. The ODT input is scanned and a case statement is executed to implement a call to the appropriate procedure to process the entered command. The program will remain in INPUT_CONTROL until a GO command is detected at which time it calls TEST_UNIT. TEST_UNIT determines what procedure to call for the selected unit and serves as a universal section driver for all tests.

INTERNAL FILES TO BOLT

In addition to the files mentioned above, BOLT also has five (5) internal files with which it works. Two of these files are used for input messages, two for output messages, and the last for diagnostic purposes. Each file is briefly explained below in the order just given.

TFILE

The file known internally to BOLT as TFILE is a random access disk input data file which contains the information necessary for the TEACH command. BOLT will, by default, look for an external file named BOLT/TEACH unless the MCP MODIFY command is used to change the default file name. The TFILE is optional and if not found, BOLT will inform the operator that the TEACH command cannot be used and continue on. To understand how the TFILE is structured, consider the following example. You wish to learn about the BYE command, so you enter and transmit to BOLT the command "TEACH BYE". BOLT reads a single header record from the TFILE which describes the information to be displayed. This record is at a predefined record location and would look something like:

```
BYE   350   12
```

where the word "BYE" is only a comment for maintenance purposes. The "350" says that the information to be displayed begins at the 350-th record in the file, and the "12" says that there are 12 records of information. BOLT then displays the 12 designated lines from the TFILE.

EFILE

The file known internally to BOLT as EFILE is a random access disk input data file which contains all the error and information messages used by the program. Every message required must have a predefined mnemonic which is equated to the record position in the EFILE of the desired messages. When an error is detected or a message is to be displayed the appropriate record is read from the EFILE and displayed (dependent on BOLT display option settings). By default, BOLT will look for an external file named BOLT/ERROR unless modified using the MCP MODIFY command. The actual structure of the EFILE is explained with an example in the section of this document named "ERROR MESSAGE FILE".

PFILE

The file known internally to BOLT as PFILE is an output message file. This file is not produced unless the PRINTER option is turned on. If the PRINTER option is turned on and there is a printer available, the information will go directly to that printer. If there is no printer available, a BACKUP.PRT file will be made. Information written to the PFILE will contain the following:

1. BOLT commands entered by the operator.

2. All information displayed by BOLT to the ODT, except the "***** ENTER NEXT COMMAND *****" message. Note PFILE information may be more detailed than that displayed to the ODT.

3. All of the TFILE information and EFILE messages.

DFILE

The file known internally to BOLT as DFILE is an output message file. This file will contain all debug and trace information for the BOLT Executive and On-Line Test Routines, should the proper program switch(es) be set during execution (see section on PROGRAM SWITCHES). The file produced will be a BACKUP.PRT file unless changed using the MCP MODIFY command. In normal use, program switches should not be set and this file, therefore, should not be produced.

DIOFILE

The internal file known to BOLT as DIOFILE is used for diagnostic IO purposes (OPEN, CLOSE, IO), and should be of no concern to BOLT users and programmers.

INTERFACING A TEST ROUTINE TO BOLT

This section explains how an On-Line test routine interfaces to BOLT, the required names of test routines, and how a test routine's initial statement should be structured.

ABOUT ON-LINE TEST ROUTINES

As explained earlier, the On-Line test routines are written separately by the programmer and then included into the BOLT source for compilation and execution by means of the SDL LIBRARY option. BOLT, at present, is capable of testing several devices. Device or unit tests are written as one procedure with each test section being a separate, nested procedure within the test itself (see FIGURE 3). The devices which BOLT is capable of testing and the required test procedure names are listed below:

1. Magnetic Tape -- MAGNETIC_TAPE_TEST
2. Card Reader -- CARD_READER_TEST
3. Line Printer -- LINE_PRINTER_TEST
4. MultiLine Control -- MULTILINE_TEST

Exactly how devices are selected for testing by BOLT, and how the aboved named procedures are called is explained in the following sections.

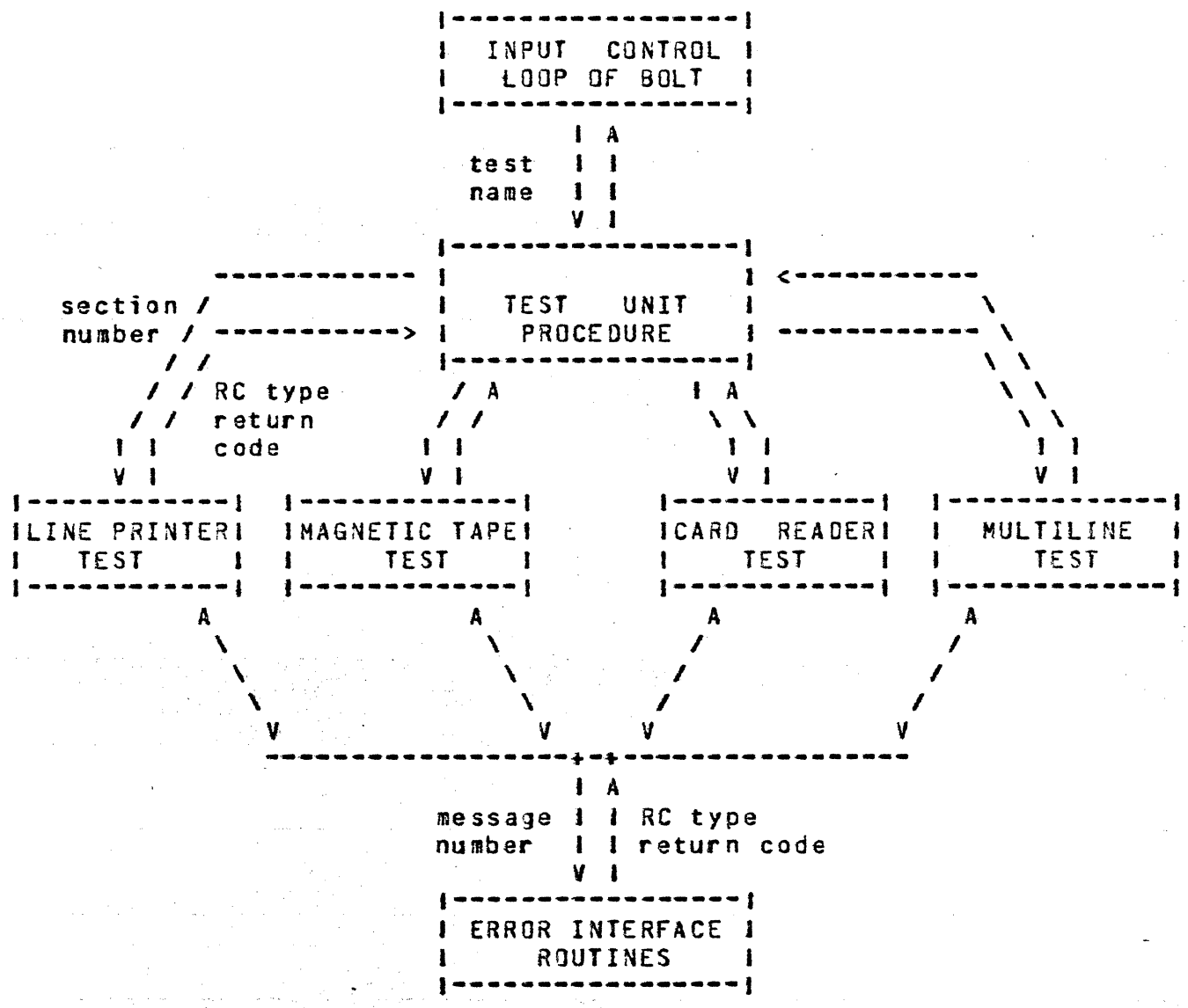


FIGURE 3 - INTERFACING A TEST ROUTINE TO BOLT. The above diagram illustrates the Top-Down structure of BOLT and indicates how the test sections interface to various parts of the executive. All of the nodes in the diagram, except the "ERROR INTERFACE ROUTINES" node, correspond to a single procedure, though that procedure may consist of many nested sub-procedures (e.g. LINE_PRINTER_TEST).

CALLING TEST ROUTINES FROM BOLT

The purpose of this section is to explain how the BOLT Executive goes about calling a test routine. The GO command tells BOLT to start (or continue) a given test. If a valid unit has been selected using the UNIT command then the GO command is accepted. BOLT then calls the test controlling procedure TEST_UNIT.

TEST_UNIT

The TEST_UNIT procedure is the controlling (driving) loop for all test routines. At its disposal is a list of test sections to be performed (result of the SECTION or UNIT command). TEST_UNIT essentially goes down the list, calling the appropriate test procedure with the current section number (obtained from the list) to be run. The test section returns a return code (see FIGURE 3) indicating the result of the task just performed. The control loop of TEST_UNIT determines the appropriate action to be taken. Return codes and the actions they cause are listed below:

RC_CONTINUE	-- normal or error free result. Go on to the next test section selected.
RC_LOCK	-- begin the current test section over again.
RC_TOO_MANY_ERRORS	-- too many errors detected for the test (see MAXIMUM command). Stop the test and go to an idle state.
RC_INPUT_PRESENT	-- input entered while the test was running has caused the test to be stopped. (CLEAR,RESTART,EXIT,UNIT).
RC_ABORT_TEST	-- the test routine has detected some abnormal condition so the test is aborted.
RC_ABORT_SECTION	-- the test secton has detected some abnormal condition so it is aborted The next section will be executed.

The return codes are futher explained in various other sections. At present, we have seen how to run a series of test sections, how BOLT goes about gaining access to a device and calling the right test routines, and how the test control loop works. Next we look at the structure of a test routine.

TEST ROUTINE STRUCTURE

The structure of an On-Line test routine is quite simple. Every test will be a single procedure, whose name has already been determined (see ABOUT ON-LINE TEST ROUTINES). This procedure will have one parameter, it being the number of the test section to be executed. Test sections are nested procedures within the test routine, the first such procedure being test section #1. The main statement of the test routine is a CASE statement, indexed off of the parameter passed to it. FIGURE 3 shows a stripped down model of what the LINE_PRINTER_TEST might look like.

To see how an actual test sections might be written (i.e. how to handle error conditions, operator input, .. etc) see the section entitled TEST ROUTINE EXAMPLE.

```
PROCEDURE LINE_PRINTER_TEST (SECTION_NUMBER)   FIXED;
FORMAL
    CURR_SECTION_NUMBER     FIXED;

% put all test constant defines here
% put all test record defines here
% put all test variable declarations here
%
% put all UTILITY ROUTINES here
%

   PROCEDURE INIT_RESTORE_TEST (INIT_RESTORE_FLAG)  FIXED;
     % initialize or restore test if necessary
   END INIT_RESTORE_TEST;

   PROCEDURE ALPHANUMERIC_RIPPLE_TEST;
     % this is section #1
   END ALPHANUMERIC_RIPPLE_TEST;

            :
            :

   PROCEDURE END_ADDRESS_TEST;
     % this is section #n
   END END_ADDRESS_TEST;

            :
            :

   CASE (CURR_SECTION_NUM);   % LP-TEST initial statement
      RETURN INIT_RESTORE_TEST (INIT_RESTORE_FLAG);
      RETURN ALPHANUMERIC_RIPPLE_TEST;       % section 1
            :                                % section 2
            :
      RETURN END_ADDRESS_TEST;               % section n
            :
   END CASE;
END LINE_PRINTER_TEST;
```

FIGURE 4 - TEST ROUTINE STRUCTURE. This figure shows the general structure of a test procedure. For a detailed example of a test routine see the section entitled "TEST ROUTINE EXAMPLE".

GLOBAL DEFINES

The following identifiers have been DEFINED in the BOLT executive and may be used by the test sections:

IDENTIFIER	VALUE	DESCRIPTION
TRUE	1	Boolean literal having the value one.
FALSE	0	Boolean literal having the value zero.
SUCCESS	1	Equivalent to TRUE. Indicates that a routine has completed without error.
FAILURE	0	Equivalent to FALSE. Indicates that a routine has completed with an error.
BOOLEAN	BIT(1)	Define for declaring boolean variables
CH	CHARACTER	Abbreviation. Used to declare character variables or display conversions.
BYTE	BIT(8)	Literal for declaring byte variables.
DEC	DECIMAL	Abbreviation. Used when converting a variable to decimal format.
CONV	CONVERT	Abbreviation. Used when converting variables to different types.
BLANK_LINE	" "	Can be used to display a blank line if necessary.
INITIALIZE	FALSE	Passed to INIT_RESTORE_TEST routine indicating that the test routine and and device should be initialized.
RESTORE	TRUE	Passed to INIT_RESTORE_TEST routine indicating that the device should be restored.

The following defines are used by the test section procedures and the test routine driver to indicate what action is to be taken. The value of the define is specified in the parentheses.

RC_CONTINUE (0)

Error Interface routines return this code to indicate that the test section is to continue. When an I/O or error check is issued and it completes successfully this return code will result. This code will also be returned if the I/O or error check completes with an exception or error condition and current error option settings indicate that the test section is to continue.

This return code is also returned to the test section driving loop should a test section complete normally.

RC_LOCK (1)

At the end of every I/O or error check, the test section will determine if the operator has selected this "lock position" to be locked on (see LOCK command). If so then the RC_LOCK return code is returned to the test section driver. The current test section will be executed again.

RC_TOO_MANY_ERRORS (2)

When an error occurs, one of the error interface routines is called to report the error and determine the action to be taken depending on the current error option settings. As errors occur, they are counted. If the ERROR_COUNT reaches the maximum error count, as set by the MAXIMUM command, then this return code is returned, indicating that too many errors have been detected while running the current test. The test section, seeing this return code, will pass it along to the test routine driver, which will in turn terminate the test.

RC_INPUT_PRESENT (3)

In order to be sensitive to unsolicited input from the operator, test sections must call the CHECK_FOR_INPUT procedure before eacn I/O or error check. If input is found, the test section is brought to an idle state, the input is parsed, and the appropriate action is taken. The test section will continue again when the GO command is entered. If, however, the operator should enter the RESTART, CLEAR, EXIT, or UNIT command then this return code will result, indicating to the test routine driver that the test is to be terminated.

This code is also returned by the CHECK_HALT_AFTER_IO routine should one of the specified commands be entered while a test section is in the idle state after an I/O operation (see HALT command).

RC_ABORT_TEST (4)

This code is returned when the test routine determines an abnormal condition for which the entire test should be aborted.

RC_ABORT_SECTION (5)

This code is returned when a test section determines an abnormal condition for which the immediate test section should be aborted. Execution will continue with the next scheduled section. (see SECTION and UNIT command).

The following codes are returned whenever the diagnostic I/O interface is called by a test section. In normal cases, test sections would expect the IO_SUCCESS return code. This is not true, however, when an I/O is issued expecting failure. Test sections check these I/O return codes to determine the appropriate action to be taken.

IO_SUCCESS (0)

Indicates that an I/O descriptor completed without exception.

IO_FAILURE (1)

Indicates that the I/O was not issued because there was an error in the format of the communicate.

IO_EXCEPTION (2)

Indicates that an I/O completed with an exception.

IO_MAX_RETRY (3)

Indicates that an I/O descriptor continued to have an exception after being retried the number of times given by the RETRY option.

IO_TIMEOUT (4)

Indicates that an I/O descriptor timed out. BOLT assumes at this point that the device is hung.

GLOBAL VARIABLES

Below is a list of variables declared by the BOLT executive which may be used by the test sections. Given is the variable name as it appears in BOLT, the variable type, and a brief description of how the variable can be used.

LOCK_POSITION **TYPE: FIXED**

This variable is used in implementing the LOCK command. It uniquely identifies major test section components such as a single IO, a loop around an IO, a data compare, ... etc. This variable must be set to one (1) at the beginning of each test section (prior to the first major component), and is bumped by one (1) before each successive test section component. Further details can be obtained from the discussion of Utility Routines and from the test routine example at the end of this document.

POSITION_LOCKED_ON **TYPE: FIXED**

This variable is used in implementing the LOCK command. If the LOCK option has been set on and an error occurs then this variable is set to the current LOCK_POSITION. At the end of each major test section component this variable must be checked to see if it is equal to the current LOCK_POSITION. If so then the return code RC_LOCK is returned to the test routine driver. (see RC_LOCK, LOCK command, and the test routine example.

ERROR_CODE **TYPE: FIXED**

This variable is set by the CHECK_ERROR_OPTIONS routine when one of the error interface routines is called. Its value will be one of the RC type return codes, indicating what action the test section should take.

ERROR_COUNT **TYPE: FIXED**

Meaningful only when test sections are currently running. Tells how many errors have occurred durint the current test. This value is displayed when the DISPLAY command is issued.

BEGIN_ERROR_COUNT **TYPE: FIXED**

Meaningful only when test sections are currently running. Tells how many errors have occurred during the current test section. This value is reset before the execution of each test section and is displayed at the end of each.

DEVICES_IN_USE **TYPE: BOOLEAN**

Indicates that a device has been successfully selected for testing (see the UNIT command). This variable will always be TRUE when viewed by a test routine.

CONTROL_SELECTED **TYPE: BOOLEAN**

Counterpart to DEVICES_IN_USE. Indicates that both a unit and a control have been selected for testing (see the UNIT command). In such a case, both variables will be TRUE.

TEST_DESC **TYPE: DIO_DESCRIPTOR**

The IO descriptor used for all IO requests through the IO Interface Routines. Any test using the interface routines for an IO will find the resulting descriptor information here. See the section named "IO DESCRIPTOR FORMAT" for the details of an IO descriptor format. This descriptor may be viewed by the operator at any time by issuing the DISP DESC command, without a descriptor index.

TEST_EXT_RSLT **TYPE: BIT (64)**

All IO descriptors are issued along with a request for an extended result descriptor, though the unit may not provide one. In such a case, this field will be zero. If the unit does provide an extended result then this field will contain that result if it was provided on the last IO issued through the IO interface routines.

TEST_DATA **TYPE: (1024) BYTE**

Just as the IO descriptor and extended result field are provided by BOLT, so is this data buffer. Test routines have the option of using this buffer for IO data or defining their own data buffer.

***** DYNAMIC MEMORY ALLOCATION *****

EXCEPTION_FOUND **TYPE: BOOLEAN**

This boolean indicates whether or not an exception occurred on the last sequence of IO descriptors issued (i.e. PCAP mode descriptors as well as TEST_DESC).

CURRENT_PORT_CHANNEL **TYPE: BIT (7)**

The PORT and CHANNEL of the unit by BOLT to be tested. Only valid if the variable DEVICES_IN_USE is TRUE. This information is automatically inserted into the IO descriptor before it is issued to the selected device.

CURRENT_UNIT_NUMBER **TYPE: BIT (4)**

The UNIT NUMBER of the unit that is currently selected by BOLT to be tested. (The units' port and channel are found in the CURRENT_PORT_CHANNEL variable). This information is automatically inserted into the IO descriptor OP_CODE field just before the IO descriptor is issued.

USE_DEFAULT_DATA **TYPE: BOOLEAN**

The operator may specify the data to be used in some test sections, overriding the default data. This flag is set to TRUE if the default data, rather than operator entered data, is to be used for the test section.

USE_DEFAULT_LENGTH **TYPE: BOOLEAN**

The operator may specify the data length to be used in some test sections, overriding the default length. This flag is set to TRUE if the default length, rather than operator entered length, is to be used for the test section.

USE_DEFAULT_RECORDS **TYPE: BOOLEAN**

The operator may specify the number of records to be used in some test sections, overriding the default records. This flag is set to TRUE if the default records, rather than operator entered records, are to be used for the test section.

RETRY_OVERRIDE **TYPE: BOOLEAN**

This boolean may be set to TRUE by test sections to override the retry capabilities of the General Diagnostic IO Routine (DO_IO), giving the test section complete control over the retry facility.

INIT_RESTORE_FLAG **TYPE: BOOLEAN**

When a unit is succussfully obtained through a Diagnostic Open for BOLT use, the INIT_RESTORE_TEST routine of the proper test routine is called with this variable set to INITAILIZE or RESTORE. INITIALIZE tells the test routine to do any program and variable initialization required for this test. RESTORE tells the test routine to restore program variables and the device back to a normal state. Note: the amount of work required is device dependent.

DIAGNOSTIC I/O INTERFACE

The MCP provides a Diagnostic IO facility which allows On-Line test routines to have "direct" access to the system peripherals using MCP communicates. The following communicates are valid:

DIAGNOSTIC OPEN

The Diagnostic Open allows the test routine to gain access to a single unit or all units on a control. Access may be exclusive only.

It should be noted that the On-Line test routine need not be concerned with the Diagnostic OPEN. When the BOLT operator enters the UNIT Command, specifying a valid unit, the OPEN on that unit is done automatically by BOLT. If for some reason the OPEN should fail, then an error message telling why is displayed and the UNIT command has no effect (see UNIT command). Refer to INTERFACING A TEST ROUTINE TO BOLT for an example.

DIAGNOSTIC IO

The Diagnostic IO communicate allows the test routine to do any valid IO to a unit that has been assigned to the program through a Diagnostic Open. The program doing the Diagnostic IO also has complete control over the error reporting and retry capabilities, normally done by the MCP. The Diagnostic IO interface has been broken up into two levels for ease of use and maintainability. Refer to "IO INTERFACE ROUTINES" below for a full explanation.

DIAGNOSTIC CLOSE

The Diagnostic CLOSE communicate releases the specified unit(s) back to MCP control. The only restriction on a Diagnostic Close is that the unit(s) must have been assigned to the program through a Diagnostic Open.

As with the Diagnostic OPEN, the On-Line test routines need not be concerned with the Diagnostic CLOSE. The CLOSE will be done automatically whenever the operator enters the BYE, RESTART, or a valid and successful UNIT Command. If for some reason the Diagnostic CLOSE should fail then an error message telling why is displayed and the respective command has no effect. This is true except for the case of the RESTART command. All program variables will have been brought back to their default state, however, BOLT will still be connected to the last unit(s) successfully obtained via the UNIT command.

IO INTERFACE ROUTINES

There are two levels to the diagnostic IO interface routines, those which the On-Line test routines see, and a general diagnostic IO routine which they do not. In most cases, the On-Line test routines will call one of the visible **"interface"** routines with the appropriate parameters which, in turn, will call the general IO routine. The requested IO is attempted and the result is passed back to the calling test routine through the interface routine. See FIGUREs 3 and 5.

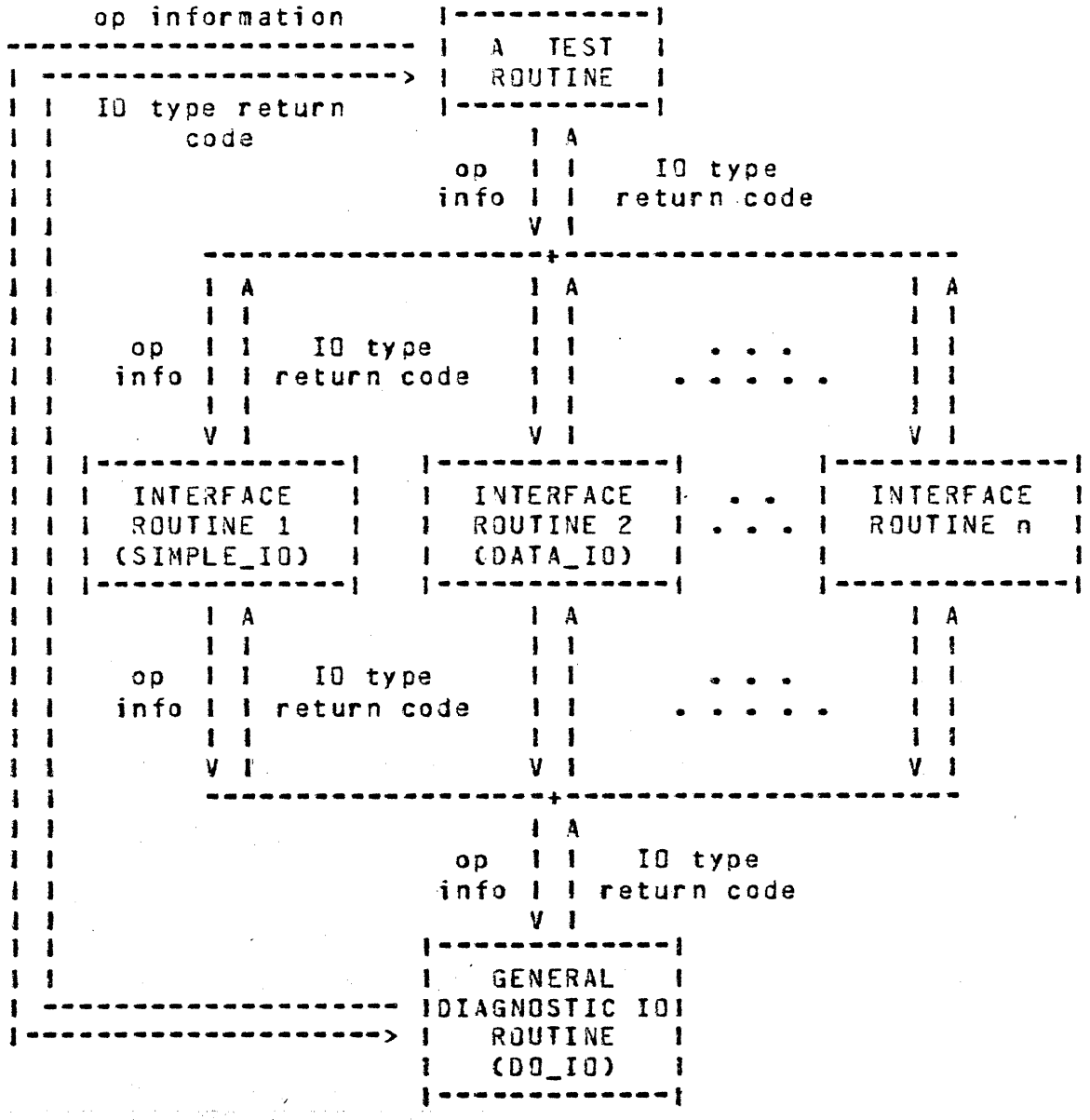


FIGURE 5 - DIAGNOSTIC IO INTERFACE. The above diagram illustrates the different levels of the diagnostic IO Interface and how these routines communicate information between one another. In special cases a test routine may call the DO_IO routine, but this is warned against since that routine assumes all descriptor fields have been correctly initialized.

GENERAL DIAGNOSTIC IO ROUTINE (DO_IO)

The general diagnostic IO routine is available to the test routines, but will usually be called indirectly by one of the second level interface routines. It has parameters for every required field and option of a diagnostic IO communicate. They are as follows:

1. File Number -- SDL Diagnostic IO file
2. Extended Result -- requesting extended result
3. Wait Complete -- wait for IO to complete or not
4. Linked Descriptors -- number of linked IO descriptors
5. IO Descriptor Address -- address of first IO descriptor
6. Extended Result Addr. -- address of first extended result

The communicate is constructed from these parameters and then sent to the MCP. The result of the IO is checked and one of the following actions will be taken.

1. If the communicate failed then IO_FAILURE is returned. (The IO was never initiated).

2. If the communicate succeeded and there was no exception on the IO then IO_SUCCESS is returned

3. If there was an exception on the IO and retries are not requested (or RETRY_OVERRIDE is TRUE) then IO_EXCEPTION is returned. Otherwise, the op is retried. If the op is retried the maximum number of times without successfully completing then IO_MAX_RETRY is returned.

It should be noted that this routine assumes that IO descriptors have been constructed correctly and that all parameters are correct.

In addition to building and issuing the Diagnostic IO communicate and returning the result of the IO back to the test routines, the general IO routine (DO_IO) has some other features which are discussed below.

TRACE DESCRIPTORS

If the TRACE DESCRIPTORS or TRACE DESCRIPTORS ON or TRACE ON command has been entered by the operator then the general IO routine (DO_IO) will display every related IO descriptor before and after it has been issued.

TRACE DATA BUFFERS

If the TRACE BUFFERS or TRACE BUFFERS ON command has been entered by the operator then the general IO routine (DO_IO) routine will display every related data buffer area before and after the IO has been issued. The format of the display is dependent on the HEX and ALPHA commands. The data areas will be displayed to the ODT and PRINTER if the respective options are turned on.

RETRY OPS

The RETRY command sets the number of times that an IO is to be retried should it complete with an exception condition. The general IO routine (DO_IO) controls the retrying or re-issuing of IO descriptors. If a sequence of one or more IO descriptors has been retried more than the number of times specified by the RETRY command then retries are stopped and an appropriate error code is returned. Should a retried IO descriptor succeed then the return code will indicate IO success, unknown to the calling routine is the fact that the IO was retried.

Test sections may override the retry facility if necessary. This is accomplished by setting the "RETRY_OVERRIDE" Boolean to TRUE. Note that the test section must then implement the RETRY facility.

Interface routines to this general IO routine are discussed below. They are designed to be a simple means for test sections to issue IO descriptors.

SIMPLE_IO

This procedure is one of the second level diagnostic IO routines. The name has been chosen because this is the simplest IO interface for the test routines. SIMPLE_IO should be called only when all of the following conditions hold:

1. No data transfer is being done. A data transfer requires the beginning and ending address of the data buffer being used for the IO.

If any one of these conditions does not hold for the IO to be performed then one of the other IO interface routines is needed.

SIMPLE_IO has only one parameter, the op code for the operation to be attempted. The executive has already obtained the port, channel, and unit number of the device to be tested (refer to UNIT command). SIMPLE_IO calls SET_UP_DIO_DESCRIPTOR, setting up the IO descriptor. Next the DO_IO routine is called and the IO is attempted. The general IO routine (DO_IO) returns the result of the IO back to this procedure. This result is passed along to the calling routine of SIMPLE_IO.

The calling routine may examine the IO result and extended result descriptors by examining the appropriate fields of the global variable "TEST_DESC" and the global variable TEST_EXT_RSLT, respectively (see NOTE below). These variables will hold the most recent IO results for a test routine, and are reset for each new IO that is issued through one of the Diagnostic IO Interface Routines.

NOTE: Extended results are always requested when doing an IO, however, not all devices return an extended result. Therefore, TEST_EXT_RSLT may have no meaning for the most recent IO. The field will be zeroed before each IO, however, to avoid any confusion.

DATA_IO

This procedure is one of the second level diagnostic IO routines. Its purpose is to provide an interface for test routines requiring data transfers on the IO. Passed to this routine will be the following parameters:

1. Operation code to be performed.

2. Begin address of buffer used in data transfer.

3. End address of buffer used in data transfer.

As in the SIMPLE_IO procedure, the diagnostic IO descriptor is constructed with the supplied information. Then the IO is attempted by calling the general IO routine (DO_IO). The result status of the IO attempt is returned to DATA_IO by this routine and is simply passed along to the calling routine.

The calling routine may examine the IO result and extended result descriptors by examining the appropriate fields of the global variable "TEST_DESC" and the global variable TEST_EXT_RSLT, respectively (see NOTE below). These variables will hold the most recent IO results for a test routine, and are reset for each new IO issued through one of the Diagnostic IO Interface Routines.

NOTE: Extended results are always requested when doing an IO, however, not all devices return an extended result. Therefore, TEST_EXT_RSLT may have no meaning for the most recent IO. The field will be zeroed before each IO to avoid any confusion.

SET_UP_DIO_DESCRIPTOR

This is a general procedure for setting up IO descriptors in preparation of a diagnostic IO. The following parameters are required.

1. IO DESCRIPTOR -- descriptor to be set up
2. OP CODE -- the op to be performed
3. BEGIN DATA ADDRESS -- begin address of buffer
4. END DATA ADDRESS -- end address of buffer

The given descriptor is constructed with the supplied parameters and control is passed back to the calling routine.

IO DESCRIPTOR FORMAT

The format of IO descriptors used by BOLT is given below. Most fields will either be set to parameters provided in IO INTERFACE ROUTINES or set by PCAP commands. Of most concern will be the "IO_RESULT" field which will contain the result status of an IO operation. The descriptor fields are as follows:

```
RECORD     DIO_DESCRIPTOR
   ACTUAL_END              BIT (24),     % *R
   IO_RESULT               BIT (24),     % *R
   IO_LINK                 BIT (24),     %    *S
   IO_OP                   BIT (24),     %    *S
   IO_BEGIN_DATA           BIT (24),     %    *S
   IO_END_DATA             BIT (24),     %    *S
   IO_DISK_ADDRESS         BIT (24),     %    *S
   IO_M_EVENTS             BIT ( 8),     %         *NA
   IO_MCP_IO               BIT (16),     %         *NA
   IO_FIB                  BIT (24),     %         *NA
   IO_FIB_LINK             BIT (24),     %         *NA
   IO_BACK_LINK            BIT (24),     %         *NA
   IO_PORT_CHANNEL         BIT ( 7);
```

Where "*R" means fields returned by an IO operation, "*S" means fields that must be or can be set by BOLT, and "*NA" means not applicable.

ERROR PROCESSING INTERFACE

This section describes the error interface routines provided by BOLT. As with the IO interface routines, there are error processing routines which the On-Line test see and don't see. Upon detecting an error, the On-Line test will call one of the higher level error routines. The general outline of each error interface routine is as follows:

1. If any one of the display options (ODT, PRINTER, or ELOG) is on then the specified error message will be read from the Error File (EFILE).

2. If ODT is set then the message read is disolayed to the ODT.

3. If PRINTER is set then the message is displayed to the line printer.

4. Depending on the type of error, other information may be displayed to the ODT, Printer, and ELOG File if those options are on.

5. All (BOLT) options pertaining to error processing will be checked and a return code indicating the action to be taken will be returned to the test routine. See the section on the CHECK_ERROR_OPTIONS routine and the section on RETURN CODES for more details.

ERROR INTERFACE ROUTINES

The following error interface routines are those which the On-Line routines have access to. Exactly which one of these routines a test routine will call upon detecting an error depends on the type of data to be displayed with the error message.

ERROR

This routine is called by an On-Line test routine when it has detected an error and the test routine only wishes to display an error message and check the error options (no result descriptor information). Passed to the ERROR_MSG routine as a parameter is the error message (equated value) to be displayed (refer to ERROR MESSAGE FILE). The procedure is constructed as follows:

```
PROCEDURE ERROR ( ERR_MSG, ERR_NUM )    FIXED;
FORMAL
    ERR_MSG      FIXED,
    ERR_NUM      FIXED;

  ERROR_MSG ( ERR_MSG, ERR_NUM );
  ELOG_MSG;
  RETURN (CHECK_ERROR_OPTIONS);
END ERROR;
```

See below for an explanation of the ERROR_MSG, ELOG_MSG, and CHECK_ERROR_OPTIONS procedures. The value returned to the calling routine reflects the current setting of the error processing options and/or any operator input.

ERROR_IO

This routine is called by an On-Line test routine when it has detected an error on an IO and the wishes to display an error message, and result descriptor information to the ELOG File (done by issuing an ELOG communicate to the MCP). This error interface routine does not support any data display. Passed to the ERROR_IO routine as a parameter is the error message (equated value) to be displayed (refer to ERROR MESSAGE FILE for a complete explanation). The procedure is written as follows:

```
PROCEDURE ERROR_IO ( ERR_MSG, ERR_NUM )    FIXED;
FORMAL
    ERR_MSG      FIXED,
    ERR_NUM      FIXED;

  ERROR_MSG (ERR_MSG, ERR_NUM);
  ELOG_MSG;
  IF ELOG              % ELOG OPTION ON
     THEN DO;
        %
        % BUILD ELOG COMMUNICATE
        % WITH DESCRIPTOR  INFO.
        %
        % ISSUE ELOG COMMUNICATE
     END;
  RETURN (CHECK_ERROR_OPTIONS);
END ERROR_IO;
```

See below for an explanation of the ERROR_MSG, ELOG_MSG, and CHECK_ERROR_OPTIONS procedures. The value returned to the calling routine reflects the current setting of the error processing options and/or any operator input.

ERROR_VAL

This procedure is called by an On-Line test routine when it has detected an error pertaining to the compare of two values. Passed to ERROR_VAL is the error message (equated value) to be displayed, and the expected and obtained values found to miscompare. The procedure works as follows:

```
PROCEDURE ERROR_VAL (ERR_MSG, ERR_NUM,
                     EXP_VALUE, OBT_VALUE)   FIXED;
FORMAL
    ERR_MSG       FIXED,
    ERR_NUM       FIXED,
    EXP_VALUE   BIT(24),
    OBT_VALUE   BIT(24);

  ERROR_MSG ( ERR_MSG, ERR_NUM );
  OUT ( "EXPECTED DATA:    "   CAT  CONVERT(EXP_VALUE, CHAR));
  OUT ( "OBTAINED DATA:    "   CAT  CONVERT(OBT_VALUE, CHAR));
  RETURN (CHECK_ERROR_OPTIONS);
END ERROR_VAL;
```

See below for an explanation of the ERROR_MSG, ELOG_MSG, and CHECK_ERROR_OPTIONS procedures. The value returned to the calling routine reflects the current setting of the error processing options and/or any operator input.

ERROR_BUF

This procedure is called when an On-Line test routine detects a compare error in two data arrays. For example, a test might wish to write to a magnetic tape then read the data written, comparing two buffers for any errors. The ERROR_BUF routine is called passing it the error message (equated value), the two data arrays and the length of the obtained data array. The ERROR_BUF procedure is structured as follows:

```
PROCEDURE ERROR_BUF (ERR_MSG, ERR_NUM, EXP_BUFF,
                     OBT_BUFF, BUFF_LENGTH)   FIXED;
FORMAL
    ERR_MSG       FIXED,
    ERR_NUM       FIXED,
    EXP_BUFF      CHARACTER (*),
    OBT_BUFF      CHARACTER (*),
    BUFF_LENGTH   FIXED;

  ERROR_MSG (ERR_MSG, ERR_NUM);
  OUT ( "EXPECTED DATA" );
  DISPLAY_BUFFER (EXP_BUFF, BUFF_LENGTH);
  OUT ( "OBTAINED DATA" );
  DISPLAY_BUFFER (OBT_BUFF, BUFF_LENGTH);
  RETURN (CHECK_ERROR_OPTIONS);
END ERROR_BUF;
```

See below for an explanation of the ERROR_MSG, ELOG_MSG, and CHECK_ERROR_OPTIONS procedures. The value returned to the calling routine reflects the current setting of the error processing options and/or any operator input. The DISPLAY_BUFFER routine is assumed to be an iterative routine, displaying BUFF_LENGTH bytes of data from the buffer passed to it.

ERROR_MSG

This procedure can be used by a test section whenever it desires to display an error message without displaying values or checking the current error option settings. The procedure checks to see if any of the BOLT display options are on. If so then the error message indicated is obtained from the separate disk error file EFILE. If the ODT option is on then the error message is displayed to the ODT. Likewise for the PRINTER option. The ERROR_MSG routine is written as follows:

```
PROCEDURE ERROR_MSG  ( ERR_MSG, ERR_NUM );
FORMAL
    ERR_MSG    FIXED,
    ERR_NUM    FIXED;

  IF (ODT  OR  PRINTER  OR  ELOG) THEN
     READ EFILE [ERR_MSG] (LAST_MSG);
  IF ODT THEN
     % DISPLAY THE FOLLOWING:
     %    DEVICE BEING TESTED -- IF ANY
     %    SECTION NUMBER      -- IF ANY
     %    ERR_NUM
     %    MESSAGE READ FROM EFILE
  IF PRINTER THEN
     % WRITE TO THE PFILE:
     %    DEVICE BEING TESTED -- IF ANY
     %    SECTION NUMBER      -- IF ANY
     %    ERR_NUM
     %    MESSAGE READ FROM EFILE
END ERROR_MSG;
```

ELOG_MSG

This procedure is called to handle all message displays to the Engineering Log (ELOG File). No parameters are required. The message to be written to the ELOG File is assumed to have already been read from the EFILE and placed in the global character array LAST_MSG.

```
PROCEDURE ELOG_MSG;
  IF ELOG  THEN
     % ZIP EM THE FOLLOWING INFORMATION
     %    DEVICE BEING TESTED -- IF ANY
     %    SECTION NUMBER      -- IF ANY
     %    ERR_NUM
     %    MESSAGE STORED IN LAST_MSG
END ELOG_MSG;
```

CHECK_ERROR_OPTIONS

All On-Line test error interface routines wishing to check the error options call this routine to check the current setting of the error option flags. Depending on these settings the appropriate action will be taken and a return code will be returned to the calling routine indicating the action to be taken by the test routine. Error options checked are HALT ERROR, LOCK, and MAXIMUM (see the respective commands). The value to be returned is recorded in the global variable "ERROR_CODE". The routine is structured as follows:

```
PROCEDURE CHECK_ERROR_OPTIONS    FIXED;

  BUMP ERROR_COUNT;                % COUNT ERRORS
  IF HALTERROR
     THEN DO;
        DISPLAY "TEST SECTION STOPPED DUE TO EXCEPTION ON I/O";
        DISPLAY "ENTER NEXT COMMAND";
        INPUT_CONTROL (TRUE);      % WAIT FOR OPERATOR INPUT
        IF EXIT OR RESTART OR CLEARED OR NEW_UNIT
           THEN
              RETURN (ERROR_CODE := RC_INPUT_PRESENT);
     END;

  IF ERROR_COUNT  EQL  MAXIMUM_VALUE
     THEN DO;
        DISPLAY "MAXIMUM NUMBER OF ERRORS REACHED";
        RETURN (ERROR_CODE := RC_TOO_MANY_ERRORS);
     END;

  IF LOCKERROR
     THEN DO;
        IF NOT LOCKED
           THEN DO;
              POSITION_LOCKED_ON := LOCK_POSITION;
              LOCKED := TRUE;
           END;
     END;
   RETURN (ERROR_CODE := RC_CONTINUE);
END CHECK_ERROR_OPTIONS;
```

MESSAGE

This routine provides a means for test routines and test sections to read messages from the EFILE. Messages may be error messages or informative messages for the operator. For instance, one Line Printer Test Section may require that the Line Printer being tested is NOT READY. The message requesting the operator to make the device NOT READY, can be stored in the EFILE. This also allows for easier maintenance of output messages. The MESSAGE procedure is written as follows:

```
PROCEDURE MESSAGE (MSG_NUM)    CHARACTER VARYING;
DEFINE
     MSG_NUM    FIXED;
DECLARE
     MSG        CHARACTER (30);

  READ EFILE [MSG_NUM] (MSG);
  % STRIP OFF TRAILING BLANKS, LEAVE ONE
  RETURN (MSG);
END MESSAGE;
```

A few sample uses of this procedure and the resulting output is shown below. The reader should notice the versatility of this routine.

```
SOURCE:  DISPLAY ( MESSAGE(PLEASE_NOT_READY) );
OUTPUT:  PLEASE NOT READY THE SELECTED LINE PRINTER.

SOURCE:  DISPLAY ( MESSAGE(WRITE_FAILED) CAT
                   CONVERT (DECIMAL (LINE_NUMBER, 3), CHARACTER));
OUTPUT:  THE WRITE OP FAILED ON WRITE NUMBER 126
```

Utility, and Informative messages are placed into the EFILE in the same way that error messages are. See the next section which explains more about error messages.

ERROR MESSAGE FILE

Error messages are found in a separate disk file, internally known to BOLT as EFILE. When the ERROR_MSG routine is called it is passed a message number. This number is the position of the appropriate error message in EFILE (zero relative). The error message is read from the file and displayed where necessary, depending on the current display option settings.

When a test is being designed and coded the programmer should recognize and record all required error messages. Once the test is ready to be compiled the programmer will have to define the error message mnemonics and enter the actual messages into the appropriate spot in the error file (EFILE). For example, if a test routine wishes to display a message indicating a parity error on a read operation the following steps would be taken:

1. DEFINE
   READ_PARITY_ERROR AS # 45 #;

2. Enter the actual message in the error file at the 45-th position, 0 relative. Looking at a portion of the error file one might see:

```
<record 43>:  INVALID KEYWORD.
<record 44>:  EXPECTING A ";" BETWEEN COMMANDS.
<record 45>:  PARITY ERROR DETECTED ON READ OP.
```

3. When an error occurs call the appropriate routine passing it the error mnemonic. Such as:
   ERROR_IO (READ_PARITY_ERROR, ERR_NUM);

An example of how an error message might be displayed to a given output device is given below. This is not necessarily the form of the display for all devices.

```
ERROR: 000012   TEST: MTA   TASK: SECTION 3
PARITY ERROR DETECTED ON READ OP.
POSSIBLE FAILURE OF INTERFACE LOGIC
EXPECTED DATA  @C1C2C3C4C5C6C7C8@
RECEIVED DATA  @C1C2C3C4FFC6C7C8@
ERRORS                  **
```

BURROUGHS CORPORATION
COMPUTER SYSTEMS GROUP
SANTA BARBARA PLANT

UTILITY PROCEDURE INTERFACE

Following are a list of miscellaneous general purpose routines that are declared in the BOLT executive and may be used by the test routine.

BOJ_SECTION_MSG

A call to this procedure should be the first statement of every test section. The procedure requires one parameter which is a character string, not to exceed 50 characters, containing the following information:

"BEGIN SECTION <#>: <section name>."

The BOJ_SECTION_MSG procedure will append the unit mnemonic of the unit being tested to the front of the supplied message. <section name> is the name of the test section that is currently being executed. An example call from a test section would be:

```
BOJ_SECTION_MSG ("BEGIN SECTION 1: END ADDRESS TEST.");
```

The procedure will output the message only if the BOJ option has been selected by the operator and the section is not being repeated. The output will go to the ODT and the PRINTER if both options have been set.

EOJ_SECTION_MSG

A call to this procedure should be near the end of every test section. The procedure requires one parameter which is a character string, not to exceed 50 characters, containing the following information:

"END SECTION <#>: <section name>."

The EOJ_SECTION_MSG procedure will append the unit mnemonic of the unit being tested to the front of the supplied message. <section name> is the name of the test section currently being executed. An example call from a test section would be:

EOJ_SECTION_MSG ("EOJ SECTION 1: END ADDRESS TEST.");

On the line following the message just described, this procedure will display the number of errors detected in the test section just completed. The messages will only be displayed if the BOJ option has been selected and the section is not being repeated. The output will go to the ODT and the PRINTER if both options have been set.

FILL_BUFFER

This procedure may be called by any test routine to fill a given buffer with a given pattern. Passed to this routine are the following parameters:

1. BUFFER -- buffer to be filled with pattern
2. BUFFER_LEN -- length of the buffer to be filled in bits
3. PATTERN -- data pattern to put into BUFFER
4. PAT_LEN -- length of the data pattern in bits

The specified pattern is repeated or truncated, if necessary, to fill the BUFFER. See the TEST ROUTINE EXAMPLE section for examples of how to use this and other utility routines.

PATTERN

This procedure is called when a test section wishes to fill a buffer with a predefined pattern. Passed to this procedure are the following parameters:

1. BUFFER -- the buffer to be filled with the pattern
2. BUFFER_LEN -- length of the buffer to be filled
3. PAT_NUM -- number of the pattern to be used

When called, this procedure checks first to make sure that the pattern specified is a valid one (refer to the DATA command for a list of valid pattern indecies). Once the pattern index has been verified, the given buffer is filled with the pattern, repeating or truncating where necessary. This procedure calls the FILL_BUFFER procedure to actually fill the BUFFER with data.

COMPARE_BUFF

A utility routine provided for COMPARE command and all test procedures for comparing two data areas. Passed to this procedure are the following parameters:

1. REC_BUF — the buffer to be compared with the standard one.
2. EXPEC_BUF — the standard buffer.
3. EXPECTED_LEN — length of the standard buffer in bits.
4. NO_OF_BITS — number of bits to be compared at a time (4 / 8).
5. DISP_FLAG — a boolean to indicate if the buffers are to be displayed in the case of data miscompare.
6. ERR_NO — error number (for test procedures).
7. RESULT_FLAG — a boolean that will be set if compare is successful.

The procedure returns a fixed value where the value returned should be checked by the test procedures in order to take an appropriate action in case of data miscompare.

When called, this procedure compares the contents of the REC_BUF with that of EXPEC_BUF by NO_OF_BITS bits at a time. It should be noted that the test procedures should check the HEX and ALPHA options and set NO_OF_BITS accordingly. If the HEX option is on, the data should be compared in four bit quantities; if the ALPHA option is set, the data should be compared in eight bit quantities; if both options are set, the data should be compared in four bit quantities.

If a data miscompare is detected, the procedure calls ERROR routine if ERR_NO is not -1; otherwise, it only informs the operator of the data mismatch. So if any of the test routines wish to check the error options and record the error message in the ELOG file, it should call this routine with ERR_NO set to anything but -1.

The procedure, if the DISP_FLAG is set, then displays 46 characters of "EXPECTED" and "OBSERVED" data to the ODT beginning at the character boundary of the data in error if the ODT option is on. It displays the entire buffers to the printer if the PRINTER option is set. The decimal position of the first HEX digit or ALPHA character is displayed at the beginning of each line in parenthesis. Compare errors are marked with asterisks (*) on the line titled "ERRORS" for each HEX digit or ALPHA character in error.

If neither ODT nor PRINTER are set, the operator is informed and buffers are not displayed.

DISPLAY_IO_RESULT

A utility routine provided for all test procedures for uniform output of IO descriptor results. Passed to this routine is the "IO_RESULT" field of the IO descriptor in question. The result field is displayed following a label identifying it.

WAIT_FOR_INPUT

This utility procedure is for test sections only. It is provided so that test sections may bring BOLT to an idle state, waiting input. For example, a test section in the LINE PRINTER TEST might want to stop temporarily and wait for the operator to ready the printer that is being tested. The WAIT_FOR_INPUT routine is a boolean procedure where the value returned means:

TRUE -- the operator entered an EXIT, CLEAR, RESTART, or UNIT command. The test procedure is to be halted.

FALSE -- the GO command was entered and the test is to continue.

A typical use of this utility routine would be as follows:

```
DISPLAY (MESSAGE ( MSG_NUM ));
IF WAIT_FOR_INPUT  THEN RETURN (RC_INPUT_PRESENT);
    :
    :
```

For this example, we will assume that MSG_NUM is a defined mnemonic equated to a numeric value; that value being the position in the Error File (EFILE) of the message "PLEASE READY THE LINE PRINTER THAT IS BEING TESTED.". The message is displayed and BOLT is brought to an idle, waiting input state. The operator should ready the line printer in question. Entering a GO command will continue the test section currently being executed.

ACCEPT_AN_ANSWER

This boolean procedure allows interaction with the user. The user is asked to supply a yes or no input to the ODT. The procedure returns a boolean according to the entry: a 1 for yes and a 0 for no. Only yes or no will be accepted. The routine loops until either is supplied.

CHECK_FOR_INPUT

CHECK_FOR_INPUT is a boolean procedure which checks to see if the operator has entered any input on the ODT. A call to this routine must be made before every test section fragment that is labeled with a "LOCK_POSITION" (see GLOBAL VARIABLES and the TEST ROUTINE EXAMPLE). The CHECK_FOR_INPUT routine returns values having the following meaning.

TRUE -- the operator entered an EXIT, CLEAR, RESTART, or UNIT command. The test procedure is to be halted.

FALSE -- the GO command was entered and the test is to continue, or no input has been entered.

A typical use of this utility routine would be as follows:

```
BUMP LOCK_POSITION;
IF CHECK_FOR_INPUT   THEN RETURN (RC_INPUT_PRESENT);
    :
    :
***  error check or IO attempt  ***
    :
    :
```

CHECK_HALT_AFTER_IO

This boolean procedure checks to see if the operator has requested BOLT to halt after each IO (see HALT command). If so, then BOLT will go to an idle, waiting input state. A call to this procedure must appear after the IO request or error check of every test section fragment that is labeled with a "LOCK_POSITION". Values returned by this procedure have the following meaning:

TRUE -- the operator entered an EXIT, CLEAR, RESTART, or UNIT command. The test procedure is to be halted.

FALSE -- the GO command was entered and the test is to continue, or the option to halt after each IO was not turned on.

A typical use of this utility routine would be as follows:

```
      :
      :
***  error check or IO attempt  ***
      :
      :
IF CHECK_HALT_AFTER_IO   THEN RETURN (RC_INPUT_PRESENT);
IF LOCK_POSITION EQL POSITION_LOCKED_ON
   THEN
      RETURN (RC_LOCK);
```

The line of code checking to see if the LOCK_POSITION is equal to the POSITION_LOCKED_ON shows how test sections implement the LOCK command. In the following section of this document a model test section fragment is shown to give the reader a better understanding of how some of these "required" utility routines are used. After that an example test routine is given.

TEST ROUTINE EXAMPLE

Following is a test routine example which demonstrates the use of some of the IO interface, ERROR interface, and utility routines of BOLT. This test routine example also shows the reader how to use return codes returned by these interface routines, and how a particular test section is called. Before looking at the test routine example, however, it is necessary to show the reader what a model test section fragment should look like. Following that several test sections are constructed from these fragments, showing the overall construction of an "example" test routine.

TEST SECTION FRAGMENT

A model test section fragment is shown in FIGURE 6 below. A test section fragment is defined as being any IO attempt or error check required in a test section. The required format of each fragment consists of the following (in the order given):

1. The global variable "LOCK_POSITION" is bumped to the error (or set to one if this is the first test section fragment in the test section). This variable is used for the lock option and for indicating error positions

2. A call to the "CHECK_FOR_INPUT" routine is made to see if the operator has entered any input.

3. The IO is attempted or the Error Check is made.

4. A call to the "CHECK_HALT_AFTER_IO" routine is made to see if BOLT is supposed to halt after the IO or Error Check.

5. The "LOCK_POSITION" and "POSITION_LOCKED_ON" variables are checked to see if BOLT is locked on this error.

Other tasks may be performed before and after any of these.

```
%************************************
%*  MODEL TEST SECTION FRAGMENT  **
%************************************
%
        :
        : possibly other test section fragments
        :

  BUMP LOCK_POSITION;
  IF CHECK_FOR_INPUT  THEN RETURN (RC_INPUT_PRESENT);

        :
        : possible code to set up of IO descriptor for
        : upcoming IO.
        :
   ***** the IO or Error Check *****
        :
        : if an  unexpected error  occurred  then call
        : of  the Error Interface  routines.  Possible
        : that the test could be halted.
        :

   IF CHECK_HALT_AFTER_IO  THEN RETURN (RC_INPUT_PRESENT);
   IF LOCK_POSITION  EQL  POSITION_LOCKED_ON
       THEN RETURN (RC_LOCK);

        :
        :
        :
```

FIGURE 6: - MODEL TEST SECTION FRAGMENT. This figure gives the essential parts for every test section fragment.

BURROUGHS CORPORATION
COMPUTER SYSTEMS GROUP
SANTA BARBARA PLANT


```
+-------------------------------------------------------------------------%
+                                                                         %
+   L      I N   N EEEEE        PPPP  RRRR  I N   N TTTTT EEEEE RRRR     %
+   L      I NN  N E            P   P R   R I NN  N   T   E     R   R    %
+   L      I N N N EEE          PPPP  RRRR  I N N N   T   EEE   RRRR     %
+   L      I N  NN E            P     R  R  I N  NN   T   E     R  R     %
+   LLLLL  I N   N EEEEE        P     R   R I N   N   T   EEEEE R   R    %
+                                                                         %
+                                                                         %
+                       TTTTT  EEEEE   SSSS   TTTTT                       %
+                         T    E      S         T                        %
+                         T    EEE     SSSS     T                        %
+                         T    E           S    T                        %
+                         T    EEEEE   SSSS     T                        %
+                                                                         %
+-------------------------------------------------------------------------%
+                                                                         %
+  PURPOSE:               THIS IS A  SAMPLE TEST PROCEDURE DESIGNED TO    %
+ ---------               SHOW  THE READER  HOW  THE  TEST  PROCEDURES    %
+                         SHOULD BE STRUCTURED, HOW THE  DIAGNOSTIC IO    %
+  AND ERROR INTERFACE ROUTINES ARE USED,  HOW BOLT UTILITY  ROUTINES     %
+  ARE USED, AND  MORE.  THE TEST CASES SHOWN MAY NOT NECESSARILY DE-     %
+  PICK ACTUAL TEST CASES.        ***** THIS IS ONLY AN EXAMPLE *****.    %
+                                                                         %
+  IN SOME PLACES,  PSUEDO-CODE IS USED.  PSUEDO-CODE  SHOULD  NOT BE     %
+  CONFUSED WITH ACTUAL CODE.                                             %
+                                                                         %
+-------------------------------------------------------------------------%
PROCEDURE LINE_PRINTER_TEST ( SECTION_NUMBER )     FIXED;
FORMAL
    SECTION_NUMBER       FIXED;
```

```
+-------------------------------------------------------------------------%
+    P R I N T E R    E R R O R    M E S S A G E    D E F I N E S    %
+-------------------------------------------------------------------------%
DEFINE
    TEST_OP_FAIL                AS          # 75 #,
    SGL_WRITE_FAIL              AS          # 76 #,
    DBL_WRITE_FAIL              AS          # 77 #,
    SECTION_ABORTED             AS          # 78 #,
    NOT_RESPONDING              AS          # 79 #,


+-------------------------------------------------------------------------%
+            O P E R A T I O N    C O D E S    U S E D                    %
+-------------------------------------------------------------------------%

    OP_TEST                     AS          # @8C0C00@ #,
    OP_SGL_WRITE                AS          # @4E0000@ #,
    OP_DBL_WRITE                AS          # @4F0000@ #;


+-------------------------------------------------------------------------%
+          V A R I A B L E S    U S E D    I N    T E S T                 %
+-------------------------------------------------------------------------%

DECLARE
    LEN                         BIT (16),               % LENGTH OF DATA FOR OP
    WRITE_DATA                  BIT (MAX_DATA_LEN),     % DATA BUFFER FOR WRITE
    IO_RESULT                   FIXED,
    ERROR_RESULT                FIXED;
```

```
+-----------------------------------------------------------------------%
+                                                                       %
+                F I L L    P R I N T E R    B U F F E R               %
+                                                                       %
+-----------------------------------------------------------------------%
+                                                                       %
+  THIS  PROCEDURE IS  CALLED  BY TEST  SECTIONS TO  FILL THEIR  DATA   %
+  BUFFER WITH  EITHER A DEFAULT DATA PATTERN  OR USER PROVIDED DATA.    %
+  PASSED  TO THIS  PROCEDURE  AS FORMAL  PARAMETERS IS THE FOLLOWING    %
+  INFORMATION:                                                         %
+                                                                       %
+     1.  BUF        -- BUFFER TO BE FILLED WITH DATA                   %
+     2.  BUF_LEN    -- LENGTH OF BUF.                                  %
+     3.  PAT        -- DEFAULT PATTERN TO BE PLACED IN BUF             %
+     4.  PAT_LEN    -- LENGTH OF THE DATA PATTERN PAT.                 %
+                                                                       %
+  IF THE "USE_DEFAULT_DATA"  FLAG IS TRUE THEN THE DEFAULT DATA PAT-   %
+  TERN IS USED.   OTHERWISE, USER ENTERED  DATA IS USED.  THE GLOBAL   %
+  UTILITY ROUTINE "FILL_BUFFER" IS USED TO ACTUALLY FILL THE BUFFER.   %
+                                                                       %
+-----------------------------------------------------------------------%

PROCEDURE FILL_PRINTER_BUFFER (BUF, BUF_LEN, PAT, PAT_LEN);
FORMAL
    BUF             BIT VARYING;
FORMAL_VALUE
    BUF_LEN         FIXED,
    PAT             BIT VARYING,
    PAT_LEN         FIXED;

IF USE_DEFAULT_DATA
  THEN
    FILL_BUFFER (BUF, BUF_LEN, PAT, PAT_LEN);       % DEFAULT DATA
  ELSE
    FILL_BUFFER (BUF, BUF_LEN, DATA_ARRAY, 256*8)   % OPERATOR DATA
+
END FILL_PRINTER_BUFFER;
```

BURROUGHS CORPORATION
COMPUTER SYSTEMS GROUP
SANTA BARBARA PLANT


```
+-------------------------------------------------------------------------%
+                                                                         %
+                          P T R     R I P P L E                          %
+ SECTION #1                                                              %
+-------------------------------------------------------------------------%
+                                                                         %
+  THIS IS A  SAMPLE  TEST SECTION.  THOUGH THERE WILL ACTUALLY BE A      %
+  "PTR_RIPPLE"  TEST SECTION IN THE LINE  PRINTER TEST,  THIS IS NOT     %
+  WHAT THE  TEST SECTION DOES.  AGAIN,  THIS IS ONLY AN EXAMPLE TEST     %
+  SECTION.                                                               %
+                                                                         %
+  SEQUENCE OF OPERATIONS:                                                %
+                                                                         %
+      1.  TEST OP TO THE PRINTER.                                        %
+                                                                         %
+      2.  SINGLE SPACE WRITE OP TO PRINTER.  DATA IS 80 CHARACTERS       %
+          OF REPEATING PATTERN "ABABAB....".                             %
+                                                                         %
+      3.  DOUBLE SPACE WRITE OP TO PRINTER. DATA IS 132 CHARACTERS       %
+          OF REPEATING PATTERN "CDCDCD....".                             %
+                                                                         %
+  THIS PROCEDURE  RETURNS ONE  OF THE RETURN  CODES DESCRIBED IN THE     %
+  "GLOBAL DEFINES" AND "TEST_UNIT" SECTIONS OF THE DOCUMENT.             %
+                                                                         %
+-------------------------------------------------------------------------%
PROCEDURE PTR_RIPPLE   FIXED;

  BOJ_SECTION_MSG ( "PRINTER: BEGIN SECTION 1: PTR_RIPPLE.");
+*****************
+**   LOCK #1   **
+*****************
+
  LOCK_POSITION := 1;
  IF CHECK_FOR_INPUT  THEN RETURN (RC_INPUT_PRESENT);
  IF SIMPLE_IO (OP_TEST)  NEQ  IO_SUCCESS                  % *** IO ***
     THEN IF ERROR_IO (TEST_OP_FAIL,1) NEQ RC_CONTINUE
        THEN RETURN (ERROR_CODE);                  % RETURN STOPPING REASON
  IF CHECK_HALT_AFTER_IO  THEN RETURN (RC_INPUT_PRESENT);
  IF LOCK_POSITION EQL POSITION_LOCKED_ON THEN RETURN (RC_LOCK);
```

```
+****************
+**   LOCK #2  **
+****************
+
  BUMP LOCK_POSITION;
  IF CHECK_FOR_INPUT  THEN RETURN (RC_INPUT_PRESENT);
  FILL_PRINTER_BUFFER (WRITE_DATA, COLUMNS, "AB", 2*CHARS);
  IF DATA_IO ( OP_SGL_WRITE,
               DATA_ADDRESS (TEST_DATA),
               DATA_ADDRESS (TEST_DATA) + LEN,
             )   NEQ   IO_SUCCESS                % WRITE OP UNSUCCESSFUL?
     THEN IF ERROR_IO (SGL_WRITE_FAIL,2)  NEQ  RC_CONTINUE
        THEN RETURN (ERROR_CODE);
  IF CHECK_HALT_AFTER_IO  THEN RETURN (RC_INPUT_PRESENT);
  IF LOCK_POSITION EQL ERROR_LOCKED_ON  THEN RETURN (RC_LOCK);


+****************
+**   LOCK #3  **
+****************
+
  BUMP LOCK_POSITION;
  IF CHECK_FOR_INPUT  THEN RETURN (RC_INPUT_PRESENT);
  FILL_PRINTER_BUFFER (WRITE_DATA, COLUMNS, "CD", 2*CHARS);
  IF DATA_IO ( OP_DBL_WRITE,
               DATA_ADDRESS (TEST_DATA),
               DATA_ADDRESS (TEST_DATA) + LEN,
             )  NEQ  IO_SUCCESS                  % WRITE IS UNSUCCESSFUL
     THEN IF ERROR_IO (DBL_WRITE_FAIL,3)  NEQ  RC_CONTINUE
        THEN RETURN (ERROR_CODE);
  IF CHECK_HALT_AFTER_IO  THEN RETURN (RC_INPUT_PRESENT);
  IF LOCK_POSITION EQL POSITION_LOCKED_ON  THEN RETURN (RC_LOCK);


  EOJ_SECTION_MSG ( "PRINTER: END   SECTION 1: PTR_RIPPLE");
  RETURN (RC_CONTINUE);
+
END PTR_RIPPLE;
```

BURROUGHS CORPORATION
COMPUTER SYSTEMS GROUP
SANTA BARBARA PLANT


```
+-------------------------------------------------------------------------%
+                                                                         %
+                          P T R     A L I G N                            %
+ SECTION #2                                                              %
+-------------------------------------------------------------------------%
+                                                                         %
+  THIS IS A  SAMPLE  TEST SECTION.   THOUGH THERE WILL ACTUALLY BE A     %
+  "PTR_ALIGN"  TEST SECTION  IN THE LINE  PRINTER TEST,  THIS IS NOT     %
+  WHAT THE  TEST SECTION DOES.  AGAIN,  THIS IS ONLY AN EXAMPLE TEST     %
+  SECTION.                                                               %
+                                                                         %
+  SEQUENCE OF OPERATIONS:                                                %
+                                                                         %
+      1.  TEST OP TO THE PRINTER.                                        %
+                                                                         %
+      2.  IF FIRST  TEST SECTION FRAGMENT (TEST OP) COMPLETES WITH       %
+          NO ERRORS THEN DO A SINGLE SPACE WRITE OP TO THE PRINTER       %
+          WITH A 132 CHARACTER LINE OF ALL H'S.  SUCH A LINE COULD       %
+          BE USED TO VERIFY CHARACTER ALIGNMENT VISUALLY. THE LINE       %
+          IS PRINTED 20 TIMES.                                           %
+                                                                         %
+  THIS PROCEDURE  RETURNS ONE  OF THE RETURN  CODES DESCRIBED IN THE     %
+  "GLOBAL DEFINES" AND "TEST_UNIT" SECTIONS OF THE DOCUMENT.             %
+                                                                         %
+-------------------------------------------------------------------------%
PROCEDURE PTR_ALIGN   FIXED;

  BOJ_SECTION_MSG ( "PRINTER: BEGIN SECTION 2: PTR_ALIGN.");
+*****************
+**   LOCK #1   **
+*****************
+  THIS SECTION IS ALMOST LIKE  LOCK #1 IN THE ABOVE TEST SECTION.
+  THIS SECTION SHOWS THE FOLLOWING:
+
+    1.  ALTERNATE WAY OF LOOKING AT RESULT OF IO
+    2.  ALTERNATE WAY OF LOOKING AT ERROR ROUTINE RESULT
+    3.  HOW ONE CAN DISPLAY IO RESULTS USING A UTILITY ROUTINE
+    4.  HOW MESSAGES CAN BE DISPLAYED BY TEST ROUTINES
+
+
  LOCK_POSITION := 1;
  IF CHECK_FOR_INPUT  THEN RETURN (RC_INPUT_PRESENT);
  IO_RESULT := SIMPLE_IO (OP_TEST);              % ***** TEST OP *****
  IF IO_RESULT NEQ IO_SUCCESS                    % LOOK AT OP RESULT
     THEN DO;
        ERROR_RESULT := ERROR_IO (TEST_OP_FAIL, 1);
        DISPLAY_IO_RESULT (TEST_DESC.IO_RESULT);
        OUT (MESSAGE(NOT_RESPONDING));          % A MESSAGE
        IF ERROR_RESULT NEQ RC_CONTINUE THEN REUTRN (ERROR_RESULT);
     END;
  IF CHECK_HALT_AFTER_IO  THEN RETURN (RC_INPUT_PRESENT);
  IF LOCK_POSITION EQL POSITION_LOCKED_ON  THEN RETURN (RC_LOCK);
```

```
+*****************
+**   LOCK #2   **
+*****************
+
+  THIS TEST SECTION DOES WHAT IS STATED IN THE PROCEDURE HEADING.
+  THIS TEST SECTION SHOWS THE FOLLOWING:
+
+    1.  HOW TO LOOK TO SEE IF THE PREVIOUS IO HAS FAILED OR NOT.
+    2.  HOW TO ABORT A TEST ROUTINE DUE TO ERRORS DETECTED.
+    3.  HOW LOOPS IN TEST SECTIONS SHOULD BE NUMBERED (LOCK_POSITION).
+    4.  HOW A TEST ROUTINE DISPLAYS AN INFORMATIVE MESSAGE.
+
+
  IF EXCEPTION_FOUND                                  % TEST OP FAILED DO NOT
     THEN DO;                                         % CONTINUE ON WITH TEST
        ERROR_MSG (SECTION_ABORTED);
        RETURN (RC_ABORT_SECTION);
     END;

  BUMP LOCK_POSITION;                                 % ERROR NUMBER 2
  LEN := COLUMNS;
  FILL_PRINTER_BUFFER (WRITE_DATA, COLUMNS, "HH", 2*CHARS);
  I := 0;
  DO H_LINE FOREVER;
     IF CHECK_FOR_INPUT  THEN RETURN (RC_INPUT_PRESENT);
     IO_RESULT := DATA_IO ( OP_SGL_WRITE,
                            DATA_ADDRESS (WRITE_DATA),
                            DATA_ADDRESS (WRITE_DATA) + LEN
                          );
     IF IO_RESULT NEQ IO_SUCCESS
        THEN IF ERROR_IO (SGL_WRITE_FAIL,2) NEQ RC_CONTINUE
           THEN DO;
              OUT ( MESSAGE(SGL_WRITE_FAIL) CAT CONV(DEC(I+1,2),CH));
              RETURN ( ERROR_CODE );
           END;
     IF CHECK_HALT_AFTER_IO  THEN RETURN (RC_INPUT_PRESENT);
     IF (BUMP I) GEQ 20 THEN UNDO H_LINE;
  END HLINE;
  IF LOCK_POSITION EQL POSITION_LOCKED_ON  THEN RETURN (RC_LOCK);


  EOJ_SECTION_MSG ( "PRINTER: END   SECTION 2: PTR_ALIGN");
  RETURN (RC_CONTINUE);
+
END PTR_ALIGN;
```

```
+-------------------------------------------------------------------------%
+                                                                         %
+                  I N I T     R E S T O R E     T E S T                  %
+                                                                         %
+-------------------------------------------------------------------------%
+                                                                         %
+  THIS PROCEDURE IS REQUIRED IN EVERY TEST ROUTINE THAT MUST DO SOME     %
+  KIND OF  INITIALIZATION  OR RESTORATION ON THE DEVICE TO BE TESTED     %
+  OR THE TEST ROUTINE ITSELF.  COMMENTS, RATHER THAN LINES OF CODE,      %
+  ARE GIVEN BELOW TO INDICATE WHAT THIS ROUTINE POSSIBLY MIGHT DO TO     %
+  TO INITIALIZE OR RESTORE PRINTER DEVICES AND THE LINE PRINTER TEST     %
+  ROUTINE.  THESE ACTIONS MAY NOT REFLECT ACTUAL TEST ACTIONS.           %
+                                                                         %
+-------------------------------------------------------------------------%
PROCEDURE INIT_RESTORE_TEST (INIT_OR_RESTORE);

  IF INIT_OR_RESTORE EQL INITIALIZE
     THEN DO INIT_PRINTER;
        %
        % GET THE CONTROL ID
        % GET THE CHARACTER SET SIZE FOR PRINTER TRAIN
        % SET GLOBAL VARIABLE "COLUMNS" TO 132 CHARACTER BITS
        %      :
        %
     END INIT_PRINTER;
+
  ELSE DO RESTORE_PRINTER;
        %
        % IF CONTROL ID INDICATES PR5/6 THEN MAKE SURE
        %    THE TRANSLATOR MEMORY IS LOADED BEFORE RETURNING
        %     :
        %
  END RESTORE_PRINTER;
END INIT_RESTORE_TEST;
```

```
+-------------------------------------------------------------------------%
+     O T H E R     P R I N T E R     T E S T     S E C T I O N S       %
+-------------------------------------------------------------------------%

           :
           :


+-------------------------------------------------------------------------%
+  M A I N    C O D E    F O R    L I N E    P R I N T E R    T E S T   %
+-------------------------------------------------------------------------%

  CASE SECTION_NUMBER;
     RETURN INIT_RESTORE_TEST;     % INITIALIZE/RESTORE TEST AND DEVICE
     RETURN PTR_RIPPLE;                 % SECTION #1
     RETURN PTR_ALIGN;                  % SECTION #2
           :                            % SECTION #3
           :
           :                            % SECTION #N
  END CASE;
END LINE_PRINTER_TEST;
```

BURROUGHS CORPORATION
COMPUTER SYSTEMS GROUP
SANTA BARBARA PLANT

INDEX