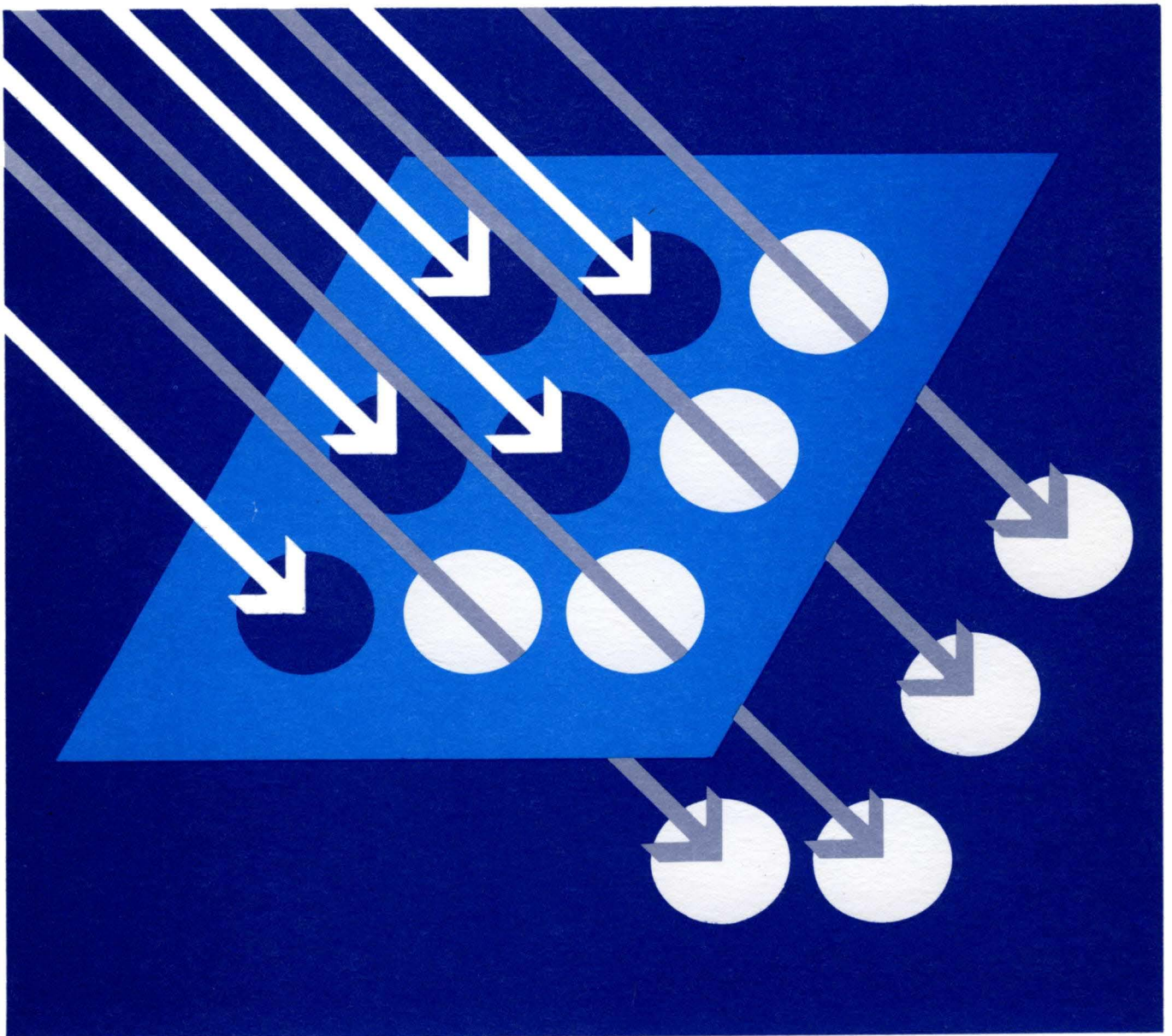


Micro Scanner Commands User's Guide

For the Sentry PLUS™ System



Copyright, 1983
by
National Computer Systems, Inc.
Minneapolis, Minnesota

*All rights reserved. No part of this book may be
reproduced in any form or by any means, without
permission in writing from the publisher.*

PREFACE

Micro Scanner Commands are designed to function with a Sentry 3000™ scanner and an IBM PC or XT with a minimum of 128K memory, one asynchronous port, and DOS 1.1 or 2.0.

This guide to Micro Scanner Commands describes the various commands and how to incorporate them into application programming. Other user manuals relating to the Sentry™ 3000 scanner are:

<u>Title</u>	<u>NCS Part Number</u>
Operator's Guide	202 151 981

The Operator's Guide provides information on the Sentry™ 3000 scanner parts, how to operate the various programs, and error messages and recovery procedure.

Installation and Maintenance Guide 202-151-999

The Installation and Maintenance Guide provides information on how to install, maintain, and repair the Sentry™ 3000 scanner.

Host Programmer's 202 151 973
Guide

The Host Programmer's Guide provides information on how to configure the scanner to enable communications with the host computer. It also describes the scanner records and explains the functions that must be performed by the host program. The Host Programmer's Guide is intended for programming without scanner commands.

TABLE OF
CONTENTS

Introduction.....iv

1: SYSTEM SETUP.....1-1
Read Technique.....1-3
Mark Discrimination.....1-7
Scanning the Forms.....1-9
Host Programming.....1-13

2: CONFIGURING THE SCANNER.....2-1
Coding the Sheet.....2-3
Configuration.....2-7

3: SCANNER COMMANDS.....3-1
Section Format.....3-3
Controlling Scanner Operations..3-7
Reconfiguring the Scanner.....3-9
Adjusting Scanner Read Level...3-13
Identifying the Form.....3-15
Scanning the Form.....3-21
Resolving the Grids.....3-25
Sending Data to the Scanner...3-41
Receiving Aux Device Data.....3-45

4: HOST PROGRAM.....4-1
Program Structure.....4-3

5: SAMPLE PROGRAM.....5-1
Program Listing (Compiled
Basic).....5-3
Program Explanation.....5-4
Interpretive Basic Listing.....5-9

6: PROGRAM ERROR CODES.....6-1

APPENDIX A: FORM PARAMETER
WORKSHEET.....A-1

APPENDIX B: TEST PROGRAM.....B-1
Compiled Basic Test Program....B-3
Interpretive Basic Test
Program.....B-19
Pascal Test Program.....B-23

APPENDIX C: PASCAL SCANDECL.PAS
FILE.....C-1

APPENDIX D: LINKING COMMANDS TO
APPLICATION PROGRAM...D-1

INTRODUCTION

This manual is presented in six sections and four appendices. The first section provides an introduction to scanner read technology and scanner commands. Section Two describes how to configure the scanner to ensure proper communications protocol. Section Three provides an overview of each command. Section Four describes host programming with scanner commands. Section Five lists and describes a sample program that incorporates scanner commands. Section Six describes program parameter error codes. Appendix A is a form parameter worksheet. Appendix B is a listing and explanation of a test program to use on your system initially. Appendix C lists the Pascal SCANDECL.PAS file, referenced by each Pascal application program. Appendix D describes how to link Scanner Commands to an application program.

1

SCANNING
OVERVIEW

Introduction.....1-2
Read Technique.....1-3
Mark Discrimination.....1-7
Scanning the Forms.....1-9
Host Programming.....1-13

INTRODUCTION

The Sentry 3000 scanner is an optical mark reader which translates marks on forms into information a computer can understand. Scanner commands allow the user to incorporate input from the scanner into application programming with a minimum of effort.

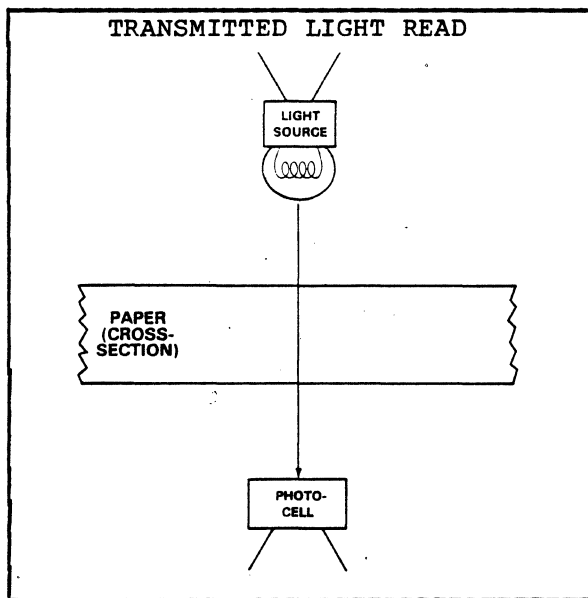
This section briefly describes how the scanner reads marks and scans forms, and how to use the scanner in a host program.

OVERVIEW: READ TECHNIQUE

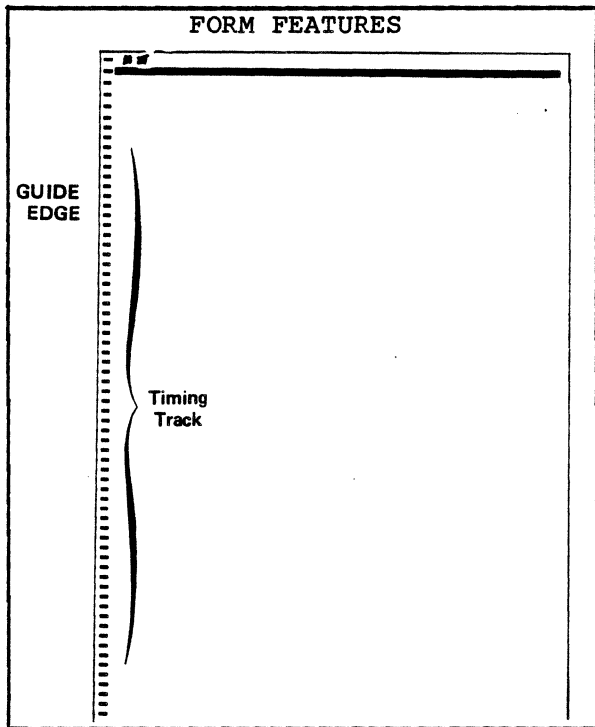
The scanner scans sheets and transmits a series of read levels to the host computer. The following paragraphs describe how the read levels are generated and how they comprise an image of the sheet.

EXPLANATION: READ TECHNIQUE

The Sentry™ 3000 scanner detects marks on forms using the "read head" which is a collection of 48 light sources above the form and 48 photocells below the form. Each of the 48 light sources beams light through the form and each photocell records the amount of light that passes through the form. In this way, the scanner reads both sides of the form at one pass through the scanner. This is called "transmitted light read."

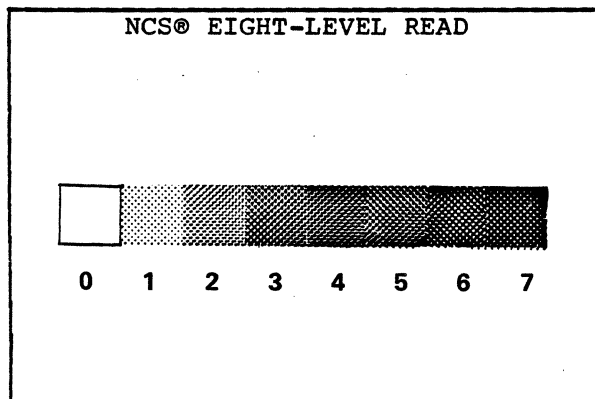


READ TECHNIQUE



EXPLANATION: TIMING MARKS

Special features of the form called "timing marks" cue the scanner's read head to read across the form. The column of timing marks is called the "timing track." The timing track is read by the photocell closest to the front of the scanner. This is why forms must always be fed into the scanner with the timing track on the left side ("guide edge") of the form.



EXPLANATION: READ LEVELS

A read level is an ASCII-coded character (0,1,2,3,4,5,6, or 7) which indicates the amount of light blocked at one position on the form. If no mark is present on the form in a particular position, the scanner will report a read level of 0 or 1. When a dark mark made with a No. 2 lead pencil is scanned, a value of 6 or 7 is assigned. Smudges and erasures will be reported by values of 1,2, or 3. Light marks will be reported as 4 or 5.

Each time the read head encounters a timing mark as the form passes through the scanner, 47 read levels plus one special value are stored in the scanner's buffer. The 47 read levels correspond to the 47 positions across an 8-1/2 inch form. The 48th value is

EXPLANATION: READ LEVELS (cont.)

the number of times that the line was scanned as the form passed under the read head.

The first read level from the form corresponds to the position on the form that is closest to the first timing mark on the leading edge of the form. The second value corresponds to the second possible response position away from the first timing mark, the third value to the third possible position, and so on.

Each time a new timing mark is read, 48 more values are added to the scanner's buffer for that form.

When the whole form has been read then the record can be transmitted to the host computer for processing.

CONSIDERATIONS: MARK READING

- 48 values are assigned even when the form is not 8-1/2" (47 response positions) wide.

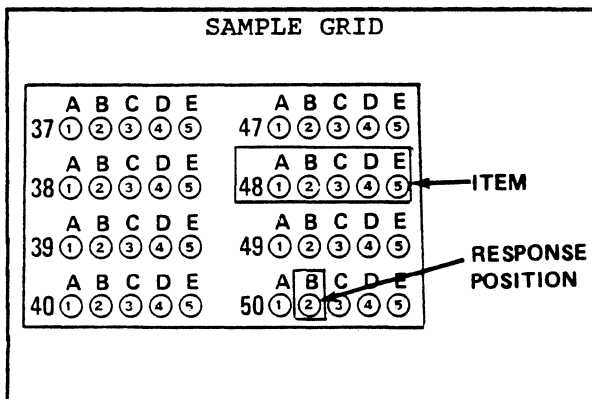
READ TECHNIQUE

OVERVIEW: MARK DISCRIMINATION

The process by which the host program determines which marks on the form are intended responses is called mark discrimination.

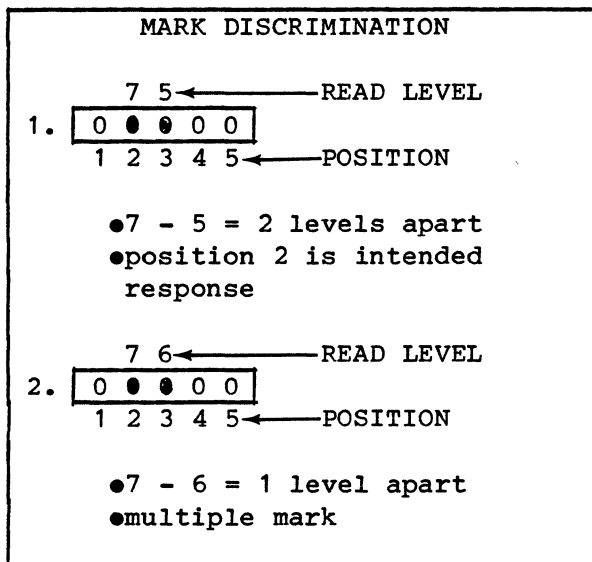
EXPLANATION: RESOLVING GRIDS

After the record of read levels is passed to the scanner, data from grids must be resolved. A single response must be selected from each grid item.



EXPLANATION: MARK DISCRIMINATION

After determining read levels the GRID command discriminates intended marks from erasures and smudges. The read levels of the responses in one item are compared and the darkest mark is selected as the intended response. However, if more than one mark is found, the read levels must be at least two read levels apart for the darkest mark to be selected as the chosen response. If read levels are not at least two levels apart, the response is deemed a "multiple" and an asterisk is placed in the appropriate position in the variable gridstr.



Consideration: Mark Threshold

While NCS recommends using a read level of 4 as the mark threshold, it is possible for the host program to establish a different threshold. One technique for determining the threshold is to take an average of the read levels on the bias bar (the bar of colored ink that extends the full width of the form). This average is the read level of paper and ink. To determine the mark threshold, add 3 or 4 to the bias bar average.

OVERVIEW: SCANNING THE FORMS

The features of the form determine the direction in which forms are fed into the scanner. Using the input tray, the operator feeds forms into the scanner one at a time. With the automatic-feed hopper, the scanner automatically picks forms one at a time from a stack of forms.

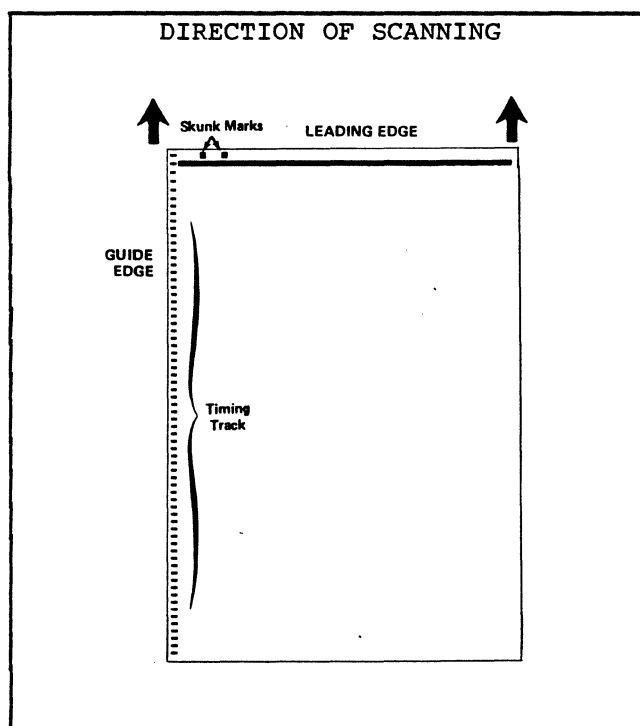
EXPLANATION: DIRECTION OF SCANNING

Two features of the form determine the direction that forms are fed into the scanner:

- Timing Track
- Skunk Marks

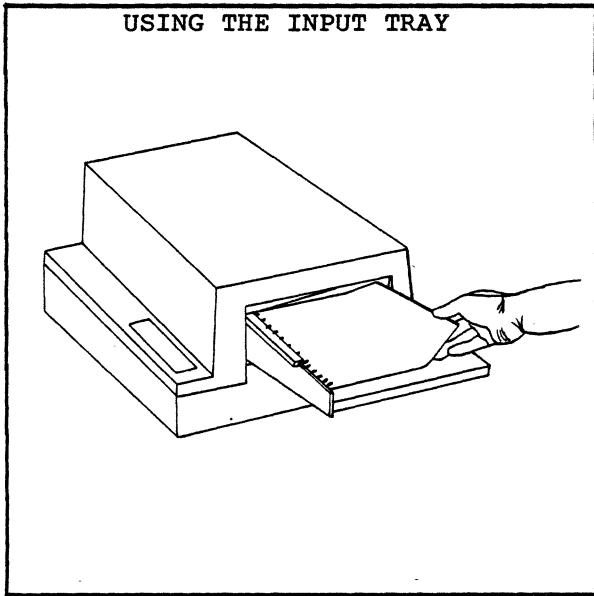
The timing track is made up of small black rectangles called timing marks. The side of the form with the timing track is placed along the guide rail of the input hopper as it is fed into the scanner. This edge is called the "guide edge".

Skunk marks are small black marks printed across from the first timing mark of a form to identify it. The edge with the skunk marks is the edge of the form that goes into the scanner first. This is called the "leading edge."



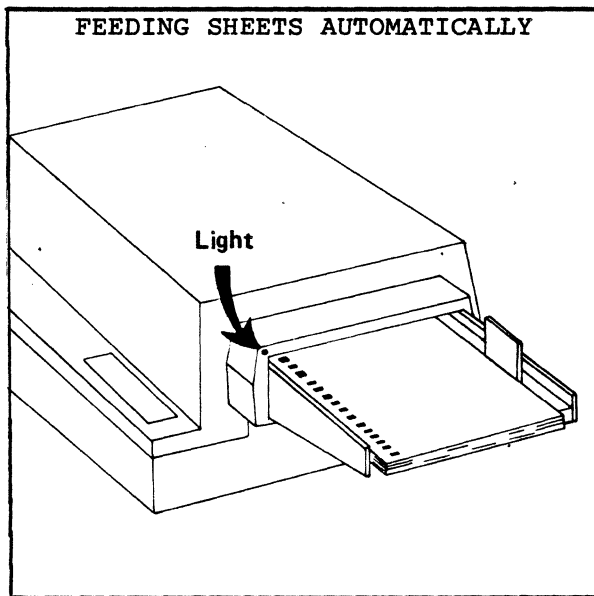
SCANNING THE FORMS

EXPLANATION: USING THE INPUT TRAY



When operating the scanner using the input tray, the operator manually slides forms forward into the scanner one at a time.

EXPLANATION: USING THE AUTO-FEED



Using the automatic-feed input hopper option, the operator prepares a uniform stack of up to 50 forms with the timing track on the left, places it in the automatic-feed input hopper against the guide rail, and lines the sheet guide up against the stack of forms. The stack is pushed forward until the green light on the hopper cover comes on. If the stack is pushed in too far, the light changes to red. To position the stack properly, pull it back until the light is green again. When the green light is on, the operator presses START. One by one the scanner picks forms from the top of the stack.

The READY light on the operator panel will go out when the auto-feed hopper becomes empty. (Pressing START when the READY light is on and sheets are present will also make the READY light go out.)

EXPLANATION: USING THE AUTO-FEED (cont.)

The READY light on the operator panel will go out when the auto-feed hopper becomes empty. (Pressing START when the READY light is on and sheets are present will also make the READY light go out.)

SCANNING THE FORMS

OVERVIEW: HOST PROGRAMMING

The scanner is used as an input device for the host program. The scanner reads forms and transfers data from forms to the host computer (IBM PC or XT). The host program must be able to control the operation of the scanner, the host and auxiliary devices, such as another microcomputer or printer. Although Scanner Commands aid in the transfer and editing of data, the host program must still process the data.

EXPLANATION: SCANNER COMMANDS

Scanner Commands are subroutines on disk which can be referenced by application programs to do data editing and provide system control, such as:

- Transfer control from the scanner to an auxiliary device or back to the scanner

- Alter communications protocol of the IBM serial interface card to communicate successfully

- Alter the read level for correct reading of especially light or dark, smudged forms

- Identify forms

- Scan forms and pass mark read levels to the application program

- Resolve marks on forms into useable data

- Transmit and receive data to and from the host and auxiliary devices

HOST PROGRAMMING

EXPLANATION: HOST PROGRAM

The host program must still determine input and output of data (whether through scanning, auxiliary devices, or from within the program), data manipulation, and incorporation of scanner commands. For a detailed description of host programming, refer to Section Four.

2

CONFIGURING THE SCANNER

Introduction.....2-2
Coding the Sheet.....2-3
Configuration.....2-7

INTRODUCTION

This section describes how to configure the scanner so that it can communicate with the micro-computer. It also describes how to define the scanner's menu.

CODING THE SHEET

<u>GRID</u>	<u>CODE</u>	<u>MANDATORY VALUES</u>	<u>USER CHANGEABLE</u>
INITIATE CODE	11, CPU	X	
POSITIVE RESPONSE	11	X	
RELEASE DOCUMENT	12	X	
NEGATIVE RESPONSE	1A	X	
SELECT AUX PORT	13	X	
SELECT SCANNER FROM HOST	14	X	
SELECT SCANNER FROM AUX	0D	X	
STOP SCANNER	0E	X	
PRINT POSITION	31	X	
PRINT DATA CODE	32	X	
AUX PORT DATA CODE	19	X	
DIGIT DATA CODE	07	X	
END OF INFO	04	X	
CHECK SUM	8	X	
START OF RECORD	leave blank	X	
END OF RECORD	0D0A	X	
END OF DOCUMENT	leave blank	X	
COMPRESS	15	X	
RECORD LENGTH	leave blank	X	
CHECK CHARACTER	leave blank	X	
AUX PORT ECHO	YES	X	
PARITY	ODD		X
STOP BITS	2		X
CHARACTER BIT LENGTH	7		X
BAUD RATE	9600		X

CONSIDERATION: USER-CHANGEABLE VALUES

The codes for parity, stop bits, character bit length, and baud rate have been specially marked to indicate that the user may change these values. The codes listed in the code column match the protocol of the host. When using an auxiliary device, all three devices (the scanner, the IBM PC, and the aux device) should be configured to match protocol.

SENTRY 3000™ ASYNCHRONOUS COMMUNICATIONS CONFIGURATOR SHEET

**SEE IMPORTANT INSTRUCTIONS
ON BACK OF FORM**

START OF RECORD										END OF RECORD										END OF DOCUMENT	COMPRESS	RECORD LENGTH					
										D A											15						
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2
3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3
4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4
5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5
6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6
7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7
A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A
B	B	B	B	B	B	B	B	B	B	B	B	B	B	B	B	B	B	B	B	B	B	B	B	B	B	B	B
C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C
D	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D
E	E	E	E	E	E	E	E	E	E	E	E	E	E	E	E	E	E	E	E	E	E	E	E	E	E	E	E
F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F

MANDATORY GRIDS

PARITY	STOP BITS	BAUD RATE
<input type="radio"/> ODD	<input type="radio"/> 1	<input type="radio"/> 110
<input type="radio"/> EVEN	<input type="radio"/> 1½	<input type="radio"/> 300
<input type="radio"/> NONE	<input checked="" type="radio"/> 2	<input type="radio"/> 600
CHARACTER BIT LENGTH		<input type="radio"/> 1200
<input checked="" type="radio"/> 7	<input type="radio"/> 8	<input type="radio"/> 2400
		<input type="radio"/> 4800
		<input checked="" type="radio"/> 9600

CHECK CHARACTER

LRC

PRINTABLE LRC

AUX PORT ECHO

YES

CAUTION STRAY MARKS IN SHADED AREAS OR ON BACK OF FORM MAY CAUSE CONFIGURATION ERRORS

INITIATE CODE	POSITIVE RESPONSE	RELEASE DOCUMENT	NEGATIVE RESPONSE	SELECT AUX PORT	SELECT SCANNER FROM HOST	SELECT SCANNER FROM AUX	STOP SCANNER	PRINT POSITION	PRINT DATA CODE	AUX PORT DATA CODE	DIGIT DATA CODE	END OF INFO
11	11	12	1A	13	14	0D	0E	31	32	19	07	04
0	0	0	0	0	0	0	0	0	0	0	0	0
1	1	1	1	1	1	1	1	1	1	1	1	1
2	2	2	2	2	2	2	2	2	2	2	2	2
3	3	3	3	3	3	3	3	3	3	3	3	3
4	4	4	4	4	4	4	4	4	4	4	4	4
5	5	5	5	5	5	5	5	5	5	5	5	5
6	6	6	6	6	6	6	6	6	6	6	6	6
7	7	7	7	7	7	7	7	7	7	7	7	7
A	A	A	A	A	A	A	A	A	A	A	A	A
B	B	B	B	B	B	B	B	B	B	B	B	B
C	C	C	C	C	C	C	C	C	C	C	C	C
D	D	D	D	D	D	D	D	D	D	D	D	D
E	E	E	E	E	E	E	E	E	E	E	E	E
F	F	F	F	F	F	F	F	F	F	F	F	F

NCS P/N 202-161-908 ©Copyright 1981, 1982, 1983, National Computer Systems, Inc.
All Rights Reserved

MANDATORY GRID CHECK SUM 0123456700

CODING THE SHEET

OVERVIEW: CONFIGURATION

Once the Sentry 3000 Asynchronous Communications Sheet has been filled out, the sheet must be scanned and the menu of system programs defined.

EXPLANATION: SCANNING THE SHEET

To scan the sheet, press and hold SEL for 10 seconds until ".c" (calibration) is displayed on the operator panel. Then, release and press SEL within 3 seconds to display ".d" (define). Press START within 3 seconds or you will have to repeat this procedure.

Feed the Sentry 3000 Asynchronous Communications Configurator Sheet into the scanner. The sheet is fed into the scanner with the timing track (column of small black rectangles) on the left side of the sheet. Be sure the left edge is completely against the guide rail of the input tray.

PROCEDURE SUMMARY

- Press and hold SEL for 10 seconds until ".c" is displayed.
- Release and press SEL within 3 seconds to display ".d".
- Press START within 3 seconds and feed the sheet. The sheet must be fed into the scanner as shown below.

COMMUNICATIONS SCANNER

<input type="radio"/> XMIT	<input type="radio"/> CTS	<input type="radio"/> READY	<input type="radio"/> ERROR
<input type="radio"/> RECV	<input type="radio"/> CD	<input type="radio"/> BUSY	<input type="radio"/> STOP

COMMUNICATIONS SCANNER

<input type="radio"/> XMIT	<input type="radio"/> CTS	<input type="radio"/> READY	<input type="radio"/> ERROR
<input type="radio"/> RECV	<input type="radio"/> CD	<input type="radio"/> BUSY	<input type="radio"/> STOP

DIRECTION INTO SCANNER

CONFIGURATION

EXPLANATION: RECONFIGURATION

Configuration parameters may be changed by marking and scanning a new Sentry 3000 Asynchronous Communications Configurator Sheet. Any of the user-changeable parameters may be changed as long as they are consistent with the requirements of the scanner, the host computer, and all other system components.

EXPLANATION: CONFIGURATION ERROR MESSAGES

ERROR MESSAGE	GRID WITH ERROR
.d00	Sheet not recognized
.d01	START OF RECORD
.d02	END OF RECORD
.d03	INITIATE CODE
.d04	INITIATE CODE (SCAN or CPU)
.d05	POSITIVE RESPONSE
.d06	RELEASE DOCUMENT
.d07	NEGATIVE RESPONSE
.d08	SELECT AUX PORT
.d09	SELECT SCANNER FROM HOST
.d0A	SELECT SCANNER FROM AUX
.d0b	STOP SCANNER
.d0C	PRINT POSITION
.d0d	PRINT DATA
.d0E	AUX PORT DATA
.d0F	DIGIT DATA
.d10	END OF INFORMATION
.d11	END OF DOCUMENT
.d12	COMPRESS
.d13	RECORD LENGTH
.d14	CHECK CHARACTER
.d15	AUX PORT ECHO
.d16	PARITY
.d17	STOP BITS
.d18	CHARACTER BIT LENGTH
.d19	BAUD RATE
.d1A	CHECK SUM

The list to the left shows the three-digit error messages that will be displayed if the coding of the specified grids is not correct. The error message will be a repeating sequence of the three digits. The message will be displayed on the scanner's operator panel when the feed bed motor stops and the ERROR light is lit.

If no error occurs when the sheet is scanned, the first menu item (.C, .A, or .S) will automatically be displayed after the sheet is read.

EXPLANATION: MENU DEFINITION

The ".d" (define) program also scans the Menu Definition Sheet. The Menu Definition Sheet determines the order in which the scanner programs are listed in the display. The scanner programs are:

- .C = Communications
- .S = Scoring
- .A = Auxiliary Device

MENU DEFINITION SHEET

1st
 2nd
 3rd
 4th
 5th

.C=Scan to Communications
 .A=Auxiliary Device
 .S=Scoring Program
 .1=First Option
 .2=Second Option

**3000
MENU
DEFINITION
SHEET**

DIRECTIONS

Mark the programs you wish and the order you want them displayed on the menu. If scanning to communications is the only program desired, mark the first response under C and leave the other two blank.

EXAMPLE: The following grid is marked so that the auxiliary device program will be the first item on the menu, the scan program second, and the scoring program third.

1st
 2nd
 3rd
 4th
 5th

.C=Scan to Communications
 .A=Auxiliary Device
 .S=Scoring Program
 .1=First Option
 .2=Second Option

NCS Trans-Ops 9830-16480-221 Copyright © 1982 National Computer Systems, Inc. All Rights Reserved. NCS 223 161 898

The sheet must be marked so that there is one and only one mark in the row labeled "1st." If a second program is desired, then there must also be one and only one mark in the row labeled "2nd," etc. In the illustration shown, the sheet has been marked so that:

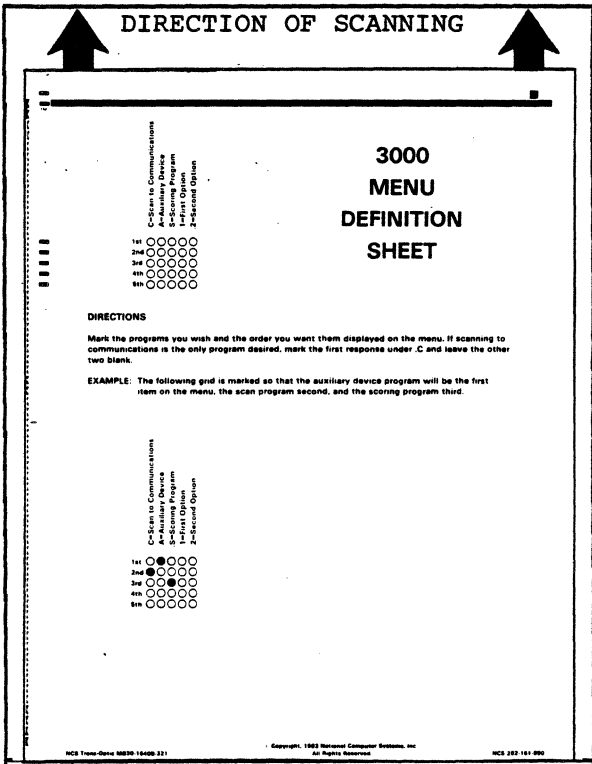
- .A = the first menu program
- .C = the second menu program
- .S = the third menu program

SAMPLE MENU DEFINITION

1st
 2nd
 3rd
 4th
 5th

.C=Scan to Communications
 .A=Auxiliary Device
 .S=Scoring Program
 .1=First Option
 .2=Second Option

CONFIGURATION



EXPLANATION: MENU DEFINITION (cont.)

The procedure for scanning the sheet is the same as for the Configurator Sheet. The orientation of the sheet as it is fed into the scanner is shown here. Be sure the left edge is completely against the guide rail of the input tray.

Once the sheet has been scanned successfully, the display will show the letter specified as "1st" on the Menu Definition Sheet. Press SEL to display the second item and press it again to display the third item (if specified on the sheet).

ERROR MESSAGE	ERROR
.d00	Sheet not recognized
.d20	Grid coded incorrectly

The error codes to the left show the three-digit messages that will be displayed if the sheet was scanned incorrectly, if the wrong sheet was scanned, or if the grid was filled out incorrectly. The error message will be a repeating sequence of the three digits. The message will be displayed on the scanner's operator panel when the feed bed motor stops and the ERROR light is lit.

CONSIDERATIONS: MENU DEFINITION

●Do not define the menu to include programs that will not be used. For example, if the scanner will be used for scoring only, define ".S" as the first and only menu item. (The scanner's scoring program is explained in detail in Section Two of the Operator's Guide.)

CONSIDERATIONS: MENU DEFINITION (cont.)

●If ".A" (auxiliary device) is defined to be the first program in the menu, the communications link between the host computer and the auxiliary device will be activated automatically when the scanner is powered ON.

●The first program in the menu is activated automatically when the scanner is powered ON.

CONFIGURATION

SCANNER
COMMANDS

Introduction.....3-2
Section Format.....3-3
Controlling Scanner
Operations.....3-7
Reconfiguring the Scanner.....3-9
Adjusting Scanner Read Level..3-13
Identifying the Form.....3-15
Scanning the Form.....3-21
Resolving the Grids.....3-25
Sending Data to Scanner.....3-41
Receiving Aux Device Data.....3-45

INTRODUCTION

The NCS Micro Scanner Commands provide the user of a 3000 scanner with the ability to incorporate data from forms into application programs on a microcomputer. By incorporating these commands into the application program, the user will be able to operate the scanner, identify documents read by the scanner, and translate data from grids on documents into strings of characters for use by the microcomputer.

The following commands will be described in this section:

CONTROL (Controlling the Scanner)
SETUP (Reconfiguring Communications Protocol)
LEVEL (Adjusting Scanner Read Level)
SKUNK (Identifying the Form)
SCAN (Scanning the Form)
GRID (Resolving the Grids)
TRANSMIT (Sending Data to System Components)
RECV (Receiving Aux Device Data)

OVERVIEW: SECTION FORMAT

Each command is described in depth, including an overview of the command, instructions on how to call each command routine, definitions of parameters, the result returned by the routine, and sample programming. For a sample program incorporating many of the commands, see Section Five. (All of the sample programs at the end of each command description in this section will be done in Compiled Basic.)

EXPLANATION: VARIABLES, CALL FORMAT

In Basic, command parameters must be initialized and then listed in parentheses within the CALL statement. The designated variable names used in this manual (Basic and Pascal) are merely examples for illustration purposes. Programmers may feel free to substitute their own choosing. However, actual data cannot be passed to the command routines. Data must be passed in variables.

In Pascal, only parameters which pass specific values to a command routine must be initialized.

CALL FORMAT - BASIC

```

20  DIM GRIDARG%(8)
30  FOR A = 0 TO 7
40  READ GRIDARG%(A):NEXT A
50  EDSTAT%=0
60  GRIDSTR$=SPACE$(200)
70  CMDERR$=SPACE$(200)
80  ARGPTR%=VARPTR (GRIDARG%(0))
90  CALL GRID% (ARGPTR%,GRIDSTR$,
              EDSTAT%,CMDERR$)
100 DATA 2,0,1,10,1,5,5,0

```

CALL FORMAT - PASCAL

```

PASCAL:
VAR
  DOCNUM,READTYPE:INTEGER;
  CMDERR:MSCSTR;
DOCNUM:= 0;
READTYPE:= 2;
STATUS:= " ";
SCAN(DOC,READTYPE,CMDERR);

```

EXPLANATION: VARIABLES, CALL FORMAT (cont.)

Basic

CALL FORMAT - BASIC	
10	DIM GRIDARG(8)
20	Y# = FRE(0) ← int. basic only
30	FOR A = 0 TO 7
40	READ GRIDARG%(A):NEXT A
50	EDSTAT%=0
60	GRIDSTR\$=SPACE\$(200)
70	CMDERR\$=SPACE\$(200)
80	ARGPTR%=VARPTR (GRIDARG%(0))
90	CALL GRID (ARGPTR%,GRIDSTR\$, EDSTAT%,CMDERR\$)
100	DATA 2,0,1,10,1,5,5,0

The Basic CALL procedure sets up and makes the command CALL. First, the micro performs house-cleaning of the string variable area with the line Y#=FRE(0). (This statement is only necessary in Interpretive Basic when using the GRID command.) Then the variables are initialized. The variable CMDERR (which lists command error numbers) and GRIDSTR (which returns grid data) are set to the maximum number of spaces which the programmer expects to receive from the command routine. Setting these variables to 200 spaces will easily ensure that all the desired data can be passed back to the application program. If an array of values is passed to the command routine, the array location must be set in the line immediately preceding the command CALL. The line ARGPTR%=VARPTR (GRIDARG%(0)) sets ARGPTR% to the location of the array GRIDARG. Then the CALL is made. The call passes the pointer to the array rather than the array itself.

EXPLANATION: ERRORS

CMDERR RETURNS:	
@	- no error
xxx	- three-digit error number
xxx xxx xxx,	etc. - error string

Scanner Commands programming errors are indicated in the variable CMDERR (command parameter error). CMDERR is a string variable which must be passed in the CALL statement to each command routine. CMDERR should be declared in Pascal and set to spaces in Basic when sent to the subroutine in the call statement. Upon return to the application program, the variable CMDERR will contain the symbol @ (ASCII 64) in the first character location, indicating no error, or a three-digit number indicating what error has

EXPLANATION: ERRORS (cont.)

occurred. If more than one error is detected, the error numbers are listed with a blank separating each three-digit number. Error codes are described in detail in Section Six of this manual.

CAUTION: CMDERR

In Pascal programming, CMDERR need only be declared before it is sent to the first command routine. In Basic, CMDERR must be initialized before each command CALL.

CONSIDERATION: SCANNER OPERATIONS

The scanner's ".C" (communications) program controls all operations of the scanner while running a program which utilizes scanner commands. Although the scanner may utilize other scanner programs during the course of application programming, Micro Scanner Commands reference those programs. The user need only be concerned that the program is started with the scanner in the ".C" program. (For descriptions of scanner programs, refer to the Operator's Guide.)

SCANNER COMMANDS

OVERVIEW: CONTROL

The CONTROL command does exactly what its name implies. It allows the user to control the operation of the scanner. The user can activate an auxiliary device (deactivating the scanner), reactivate the scanner, stop the scanner, and release documents which have been scanned. The user is able to control the scanner by inserting the CONTROL command into the application program in the following format.

EXPLANATION: CALL FORMAT

The application program calls the CONTROL routine and must pass two parameters:

- Ctrllopt
- Cmderr

One parameter is passed back to the application program:

- Cmderr

Ctrllopt

Ctrllopt is a one-digit integer variable (1,2,3, or 4) indicating which scanner control option is selected. The control options allow the user to control the following operations:
 (1) release a document from the scanner, (2) stop the scanner, (3) select (choose to operate) the auxiliary device, and (4) select (choose to operate) the scanner.

CALL FORMAT
<pre> BASIC: CTRLLOPT%=1 CMDERR\$=SPACE\$(200) CALL CONTROL%(CTRLLOPT%,CMDERR\$) PASCAL: VAR CMDERR:MSCSTR; CTRLLOPT:INTEGER; CTRLLOPT:= 1; CONTROL(CTRLLOPT,CMDERR); </pre>

CTRLLOPT OPTIONS
<pre> 1 - release document 2 - stop scanner 3 - select auxiliary device 4 - select scanner </pre>

CONTROLLING SCANNER OPERATIONS: CONTROL

Cmderr

Cmderr is a string variable which returns an error code to the application program from the CONTROL routine.

CONSIDERATION: STOPPING THE SCANNER

If the program stops the scanner, the operator must press START on the scanner in order to return control back to the program.

EXPLANATION: SAMPLE PROGRAMMING

SAMPLE PROGRAMMING

```
80 CMDERR$=SPACE$(200)
90 CTRLOPT%=3
100 CALL CONTROL%(CTRLOPT%,
                CMDERR$)
110 IF ASC(CMDERR$)<>64 THEN GOTO
    500
120 program main body...
500 CONTROL error section...
```

This program section allows the host to receive large amounts of data from the auxiliary device by selecting CONTROL option 3 - auxiliary device.

Line 80 initializes CMDERR\$. Line 90 sets CTRLOPT% to 3, which selects the auxiliary device. Line 100 makes the call to the CONTROL routine and control is passed to the auxiliary device. If the first character of CMDERR% is not "@" (ASCII 64), a CALL error has been made and a CONTROL error check is performed in lines 500 and beyond. Lines 120, etc., comprise the program body.

OVERVIEW: SETUP

The user must use the 3000 Asynchronous Communications Configurator Sheet to configure the scanner initially (see Section Two). The protocol used by the host (microcomputer) to communicate with the scanner and auxiliary device can be programmed by using the SETUP command. The protocol of the host must match the protocol of the scanner and auxiliary device if they are to communicate successfully. Since different devices can operate at different speeds and with different parameters, the user should check the operations user manual for each auxiliary device to ensure that communications protocol is correct. (For a more detailed description of communications protocol parameters, refer to the Host Programmer's Guide.)

The default values for communications protocol are:

- Baud Rate = 9600
- Parity = odd
- Data Bits = 7
- Stop Bits = 2
- Port Selection = 1

RECONFIGURING THE COMMUNICATIONS PROTOCOL: SETUP

CALL FORMAT
<pre> BASIC: BAUD%=9600 PARITY%=ASC('0') DATABITS%=7 STOPBITS%=2 PORTSEL%=1 CMDERR\$=SPACE\$(200) CALL SETUP%(BAUD%,PARITY%,DATA- BITS%,STOPBITS%,PORTSEL%, CMDERR\$) PASCAL: VAR BAUD,PARITY,DATABITS,STOPBITS, PORTSEL:INTEGER; CMDERR:MSCSTR; BAUD:= 9600; PARITY:= ORD('0'); DATABITS:= 7; STOPBITS:= 2; PORTSEL:= 1; SETUP (BAUD,PARITY,DATABITS,STOP- BITS,PORTSEL,CMDERR); </pre>

EXPLANATION: CALL FORMAT

The application program calls the SETUP routine and must pass six parameters:

- Baud Rate
- Parity
- Data Bits
- Stop Bits
- Port Selection (Portsel)
- Cmderr

One parameter is passed back to the application program:

- Cmderr

BAUD RATES
110
300
600
1200
2400
4800
9600

Baud

The variable baud refers to the baud rate, which is the rate of transmission in number of signal events per second. NCS defines baud rate as bits of binary data per second. The user must configure the host so that the baud rate matches that of the scanner (when used).

PARITY	
	<u>ASCII Value</u>
O for odd parity	79
E for even parity	69
N for no parity	78

Parity

Parity is the ASCII value of the one character constant specifying the type of parity used. Each data character transmitted or received by the scanner, host, or auxiliary device may be accompanied by a parity bit which provides a means of insuring the integrity of the character transmission. The parity bit is attached to the upper end of the seven or eight bits that represent the character.

Databits

The variable databits is an integer constant indicating the number of bits used to transmit a data character. This number does NOT include the parity bit or stop bits and only applies to data characters. The variable databits may be either 7 or 8.

DATABITS	
7	for 7 bits in one character
8	for 8 bits in one character

Stopbits

The variable stopbits is an integer constant indicating the number of stop bits used to terminate the transmission of each character. Valid values for stop bits are 1 or 2.

STOPBITS	
1	
2	

Portsel

The variable portsel is an integer variable indicating which serial port is being used. The serial port will generally be 1 unless there are two asynchronous serial channels installed on the host computer. Then portsel will be 1 or 2 depending on the address of the channel being used for the scanner.

Cmderr

Cmderr is a string variable which returns an error code to the application program from the SETUP routine.

RECONFIGURING THE COMMUNICATIONS PROTOCOL: SETUP

EXPLANATION: SAMPLE PROGRAMMING

SAMPLE PROGRAMMING

```
50 BAUD%=9600
60 PARITY%=ASC ('O')
70 DATABITS%= 7
80 STOPBITS%= 2
90 PORTSEL%= 1
100 CMDERR$ = SPACE$(200)
110 CALL SETUP%(BAUD%,PARITY%,
                DATABITS%,STOPBITS%,
                PORTSEL%,CMDERR$)
```

This program section illustrates how to initialize variables and call the SETUP routine. Lines 50 through 100 initialize the necessary parameters (baud rate, parity, data bits, stop bits, and cmderr.) Line 110 makes the SETUP routine call.

OVERVIEW: LEVEL

The scanner read level threshold is initially set by the system at 4. All marks at a read level of 4 and above are considered valid marks; those below 4 are not valid marks. The user can adjust this up or down through the use of the LEVEL command. This adjustment should only be used on small batches of problem documents (such as forms in which all marks have been made too light). (Refer to Section One for a description of how the scanner "reads" forms and discriminates between intended marks and erasures.)

EXPLANATION: CALL FORMAT

The application program calls the LEVEL routine and must pass two parameters:

- Offset
- Cmderr

One parameter is passed back to the application program:

- Cmderr

Offset

Offset is a one-digit variable (-2, -1, 0, 1, or 2). The read level is initially set by the system at 4. The variable offset represents the amount up or down the user adjusts the read level for a specific batch of forms. For example, if a user wants to scan a batch of forms with extremely light marks, the read level could be lowered by two levels (OFFSET= -2) so that the scanner would accept read levels of 2 and above as valid marks. Setting offset to zero resets the read level to its original value of 4.

CALL FORMAT
<pre> BASIC: Y#=FRE(0) OFFSET%=2 CMDERR\$=SPACE\$(200) CALL LEVEL%(OFFSET%,CMDERR\$) PASCAL: VAR CMDERR:MSCSTR; OFFSET:INTEGER; OFFSET:= 2 LEVEL(OFFSET,CMDERR); </pre>

OFFSET
<pre> read level threshold set at 4 to lower: OFFSET = -1, -2 to raise: OFFSET = 1, 2 </pre>

ADJUSTING SCANNER READ LEVEL: LEVEL

Cmderr

Cmderr is a string variable which returns an error code to the application program from the LEVEL routine when an incorrect variable has been entered in offset. If no errors occur, cmderr will return the symbol @ and the current value of the read level (as adjusted). For example, if the read level were lowered by 1, cmderr would return @3.

CONSIDERATION: READ LEVEL

Once the standard scanner read level (set by system to 4) has been altered and the affected forms have been scanned, the read level must be reset to read at its normal level. This is done by setting offset to 0 and calling the LEVEL routine.

EXPLANATION: SAMPLE PROGRAMMING

This program section raises the read level threshold to scan a batch of smudged forms. The read level is raised so that only the darkest marks (the intended responses) are read. The offset value must be set and then the LEVEL routine can be called.

Line 40 sets OFFSET% to 2 to raise the read threshold level by 2 levels. Line 50 initializes CMDERR\$. Line 60 makes the call to the LEVEL routine. If "@" (ASCII 64) does not appear in the first character of CMDERR\$, a call error has been made and the LEVEL call error check is made in lines 400, etc. Otherwise, forms are scanned and the program manipulates the data (line 80, etc.).

Lines 500 and 510 reset the read level threshold to its normal level.

```
TO RESET LEVEL

BASIC:
  OFFSET%= 0
  CMDERR$= SPACE$(200)
  CALL LEVEL%(OFFSET%,CMDERR$)
PASCAL:
  VAR
    CMDERR:MSCSTR;
    OFFSET:INTEGER;
  OFFSET:= 0;
  LEVEL (OFFSET,CMDERR);
```

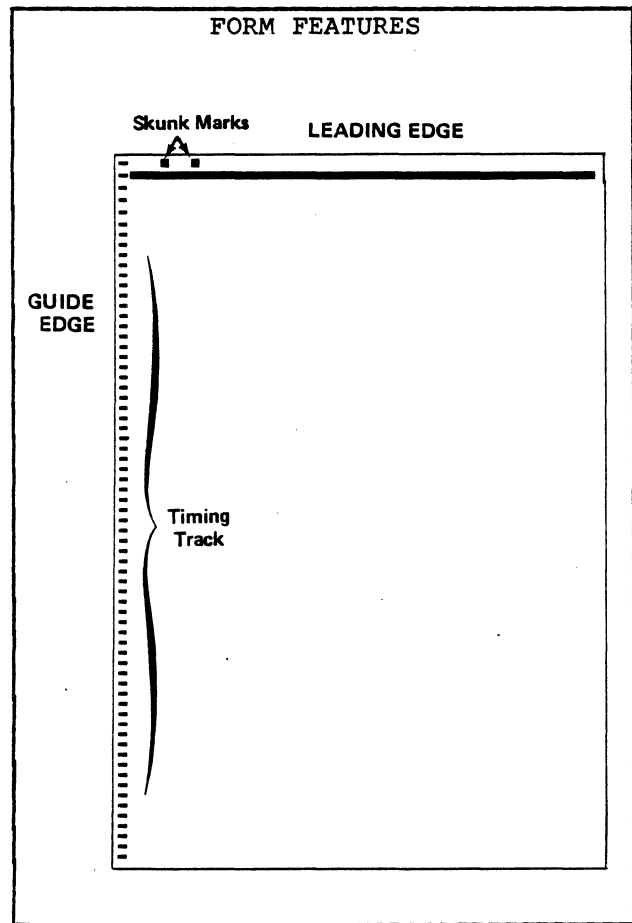
```
SAMPLE PROGRAMMING

40 OFFSET% = 2
50 CMDERR$ = SPACE$(200)
60 CALL LEVEL (OFFSET%,CMDERR$)
70 IF ASC(CMDERR%)<>64 THEN GOTO
  400
80 form scanning and data
  manipulation...
400 LEVEL error section...
500 OFFSET% = 0
510 CALL LEVEL (OFFSET%,STATUS$)
```

OVERVIEW: SKUNK

The SKUNK command allows the user to define forms which will be scanned using the SCAN command. This is done using the marks, called "skunk marks", that are located at the leading edge of the form. Each form will have a different configuration of skunk marks. Therefore, scanner commands can verify if the form is the one it expects by matching the unique skunk mark configuration.

The form definitions are saved by Micro Scanner Commands in an internal table. Then the form definitions are used by the SCAN command to ensure that the correct form is being read and that the form is being scanned properly. For example, if a form is defined in the SKUNK command and a different form is scanned in the SCAN command, an error code will be passed back to the application program. Also, if a form is scanned backwards, the SKUNK definition will not be matched and an error code will be passed back to the application program. The values in the SKUNK command are also used by the GRID command for validity checks on form parameters.



CALL FORMAT

```

BASIC:
DOCNUM%=3
CELLS%=47
TRACKS%=99
NUMMARKS%=2
MARKS%(0)=2
MARKS%(1)=4
CMDERR$=SPACE$(200)
MARKPTR%=VARPTR(MARKS%(0))
CALL SKUNK%(DOCNUM%,CELLS%,TRACKS%,
            NUMMARKS%,MARKTPR%,CMDERR$)

PASCAL:
VAR
    DOCNUM,CELLS,TRACKS,NUMMARKS:
    INTEGER;
    MARKS:SKARAY;
    CMDERR:MSCSTR;
DOCNUM:= 3;
CELLS:= 47;
TRACKS:= 99;
NUMMARKS:= 2;
MARKS[0]:= 2;
MARKS[1]:= 4;
CMDERR:= " ";
SKUNK(DOCNUM,CELLS,TRACKS,NUMMARKS,
      MARKS,CMDERR);
    
```

EXPLANATION: CALL FORMAT

The application program calls the SKUNK routine and must pass six parameters:

- Docnum
- Cells
- Tracks
- Nummarks
- Marks
- Cmderr

One parameter is returned to the application program:

- Cmderr

Docnum

Docnum is a number from 1 to 99 which is assigned to a specific form. A maximum of ninety-nine form definitions can be stored in the skunk table at any one time.

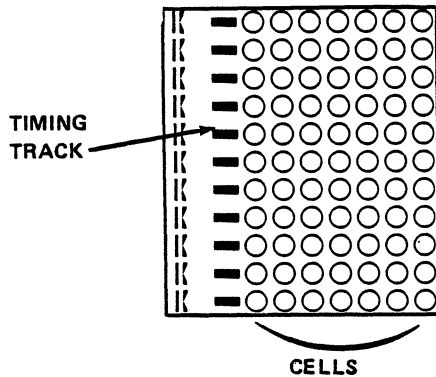
Cells

Cells is a one- or two-digit variable which specifies the number of response positions running horizontally across a form from the timing marks to the outside edge. Cells cannot be less than 1 or greater than 47.

Tracks

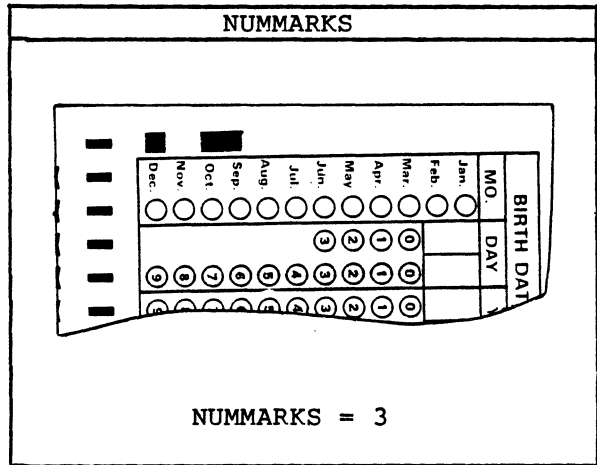
Tracks is a one- or two-digit variable which specifies the number of timing marks on the form. Timing marks are the small black rectangular marks on the left side of a form. These marks signal the scanner to read across the form at that spot. Tracks cannot be less than 1 or greater than 99.

CELLS/TRACKS



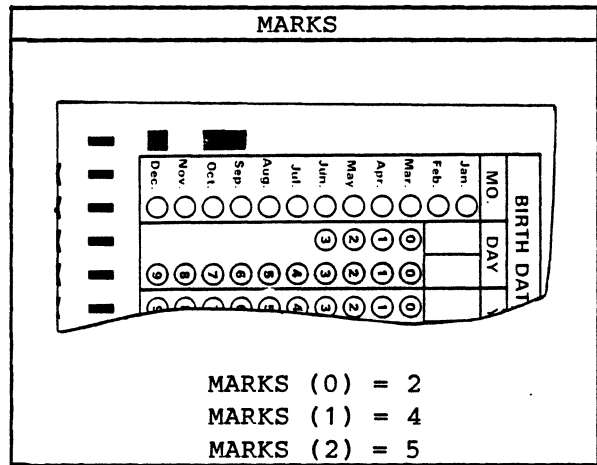
Nummarks

Nummarks is a one- or two-digit variable identifying the number of skunk mark positions which are occupied on a form. Skunk marks are black marks which occupy response positions on timing mark one of the document. Caution: nummarks is not the number of skunk marks on a form but the number of skunk mark locations occupied on a form. Although the accompanying example contains two skunk marks, they occupy three spaces. Therefore nummarks = 3. Nummarks cannot be less than 1 or greater than 47.



Marks()

Marks is an array of number values indicating the location of occupied skunk mark positions. The first position to the right of the timing mark is position 1, the second is position 2, and so on. Only one position is required to be filled. Values can vary from 1 to 47. In the example to the right, positions 2, 4, and 5 are filled. Therefore, the array is as follows: Marks (0)=2, Marks (1)=4, Marks(2)=5.



The number of individual marks should equal the value of nummarks. For example, if nummarks = 7, indicating that skunk marks occupy 7 positions, there must be 7 mark values. Since scanner commands do not directly check for the possibility that an incorrect number of marks are entered, the application program should do this.

In Basic, the pointer to the array marks is passed to the SKUNK routine. The pointer must be set to the starting location of the array. This must be done immediately preceding the SKUNK call.

Cmderr

Cmderr is a string variable which returns an error code to the application program from the SKUNK routine.

CONSIDERATIONS: SKUNK

Documents must be defined using the SKUNK command before the user can process them. Skunk commands may be issued for previously defined forms. The latest definition will be used.

REMOVING A DOCUMENT

```
BASIC:
DIM MARKS%(48)
DOCNUM%=3
CELLS%=-1
TRACKS=1
NUMMARKS%=1
MARKS%(0)=1
CMDERR$=SPACE$(200)
MARKPTR%=VARPTR(MARKS%(0))
CALL SKUNK$(DOCNUM%,CELLS%,TRACKS%,
            NUMMARKS%,MARKPTR%,CMDERR$)
PASCAL:
VAR
    DOCNUM,CELLS,TRACKS,NUMMARKS:
        INTEGER;
    MARKS:SKARAY;
    CMDERR:MSCSTR;
DOCNUM: = 3;
CELLS: = -1;
TRACKS: = 1;
NUMMARKS: = 1;
MARKS[0]: = 1;
CMDERR: = " ";
SKUNK (DOCNUM,CELLS,TRACKS,
        NUMMARKS,MARKS,CMDERR);
```

EXPLANATION: TO REMOVE A DOCUMENT

A programmer may wish to remove a document definition from the SKUNK table if a defined form will no longer be used or if a form has been incorrectly defined. The accompanying example describes how to remove a document definition from the skunk table. In this example, the definition of document 3 is removed. When removing a document from the skunk table, the variable docnum should contain the number of the document to be removed. Cells must be equal to -1. The variables tracks, nummarks, and marks may vary but they must have been initialized at some point in the program. Once the parameters have been initialized as described, the SKUNK command call is made.

EXPLANATION: SAMPLE PROGRAMMING

This program section defines the form to be scanned and stores the form definition in the SKUNK table. Then a form is scanned. If the form is the correct form, the program can perform the desired operations. If not, an appropriate error message is printed.

Line 20 initializes CMDERR\$. Lines 30, 40 and 500 comprise READ and DATA statements which read in the parameters of form number 7. Line 50 sets the pointer location for the array MARKS. Line 60 calls the SKUNK routine and stores the definition of form 7 in the skunk table.

Lines 70 through 100 initialize SCAN parameters and call the SCAN routine. The SCAN command allows the user to scan forms and transmit data. First, the SCAN parameters are initialized. READTYPE% = 2 means that a new form will be scanned. DOC% is set to 0. Since the program is written in Basic, CMDERR\$ must be reinitialized for each command call. Line 100 calls the SCAN routine and the form is scanned.

Line 110 ensures that the correct form was scanned. The form scanned will match a specific document in the skunk table, if it exists. If a document match is found in the skunk table, the value passed back in DOC% will be the number of the matching document. If no match is found, the value returned in DOC% will be 0. If the document number is 7, then the program can perform the necessary manipulations. If DOC% does not equal 7, line 110 directs the program to line 200 which prints out an error message.

SAMPLE PROGRAMMING

```

20 CMDERR$ = SPACE$(200)
30 READ DOCNUM%,CELLS%,TRACKS%,
   NUMMARKS%
40 FOR A=0 TO 2:READ MARKS%(A):
   NEXT A
50 MARKPTR% = VARPTR(MARKS%(0))
60 CALL SKUNK%(DOCNUM%,CELLS%,
   TRACKS%,NUMMARKS%,MARKPTR%,
   CMDERR$)
70 READTYPE% = 2
80 DOC% = 0
90 CMDERR$ = SPACE$(200)
100 CALL SCAN%(DOC%,READTYPE%,
   CMDERR$)
110 IF DOC%<> = 7 THEN GOTO 200
120 ELSE...
200 PRINT "WRONG FORM SCANNED"
500 DATA 7,47,63,3,1,2,7

```

IDENTIFYING THE FORM: SKUNK

OVERVIEW: SCAN

The SCAN command is capable of two useful functions: 1) It allows the user to tell the scanner to scan a sheet and pass the data from the scanner to the microcomputer, 2) it can also tell the scanner to retransmit the current data record to the microcomputer in the case of a communications problem. By using this command in the application program, the user has control over scanning and record transmission.

EXPLANATION: CALL FORMAT

The application program calls the SCAN routine and must pass three parameters:

- Doc
- Readtype
- Cmderr

Two parameters are passed back to the application program:

- Doc
- Cmderr

Doc

Doc is a number from 1 to 99 which is returned by the SCAN routine. If a document match is found in the SKUNK table, the value passed back to the application program in doc will be the number of the matching document. If no match is found, the value returned in doc will be 0. The programmer must initialize doc to 0 before calling the SCAN routine.

CALL FORMAT	
BASIC:	DOC%=0 READTYPE%=2 CMDERR\$=SPACE\$(200) CALL SCAN%(DOC%,READTYPE%, CMDERR\$)
PASCAL:	VAR DOC, READTYPE: INTEGER; CMDERR: MSCSTR; READTYPE:=2; SCAN(DOC, READTYPE, CMDERR);

CALLING OPTIONS

READTYPE =

- 2 - to request a new document from the scanner
- 3 - to request a retransmission of the current record from the scanner

Readtype

Readtype is a numeric variable identifying which calling option the user chooses. The user can choose one of two options: readtype = 2 to request a new document from the scanner, in which case a document will be scanned and its record will be passed; or readtype = 3 to request a retransmission of the current record. A retransmission should be requested only after receipt of a defective transmission.

Cmderr

Cmderr is a string variable which returns an error code to the application program from the SCAN routine.

EXPLANATION: SAMPLE PROGRAMMING

SAMPLE PROGRAMMING

```
40 FOR A = 1 TO 10
50 DOC% = 0
60 READTYPE% = 2
70 CMDERR$ = SPACE$(200)
80 CALL SCAN%(DOC%,READTYPE%,
      CMDERR$)
90 IF ASC (CMDERR$)<>64 THEN GOTO
  500
100 program main body...
480 NEXT A
490 GOTO 700
500 SCAN error check...
700 END
```

This program section scans 10 forms and ensures that the SCAN call is made correctly.

Since 10 forms will be scanned, a loop is set up to perform the scanning and data manipulation operations (lines 40-480). Lines 50-70 initialize the SCAN parameters. DOC% is initialized. In order to select the scan a sheet option, READTYPE% is set to 2. CMDERR\$ is initialized.

The call to the SCAN routine is made in line 80 and a form is scanned. If the first character of CMDERR\$ is not "@" (ASCII 64) upon return to the application program, a call parameter error has been made and the SCAN error check is made beginning on line 500. Otherwise the program performs the necessary data manipulations (lines 100, etc.). Line 490 allows for bypassing of the SCAN error section if no call error has been made and line 700 ends the program.

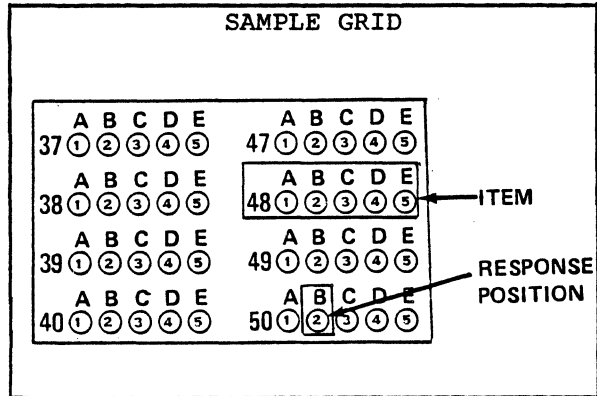
CONSIDERATIONS: SCAN

The SCAN command should not be utilized unless the document being scanned has been defined through the SKUNK command. The SCAN command utilizes data from the SKUNK form definitions. This is to ensure that the correct forms are being scanned and that forms are being scanned in the proper manner (refer to Section Three, SKUNK command). If the SCAN command is used without a previous SKUNK command, a record will still be passed to the microcomputer but an error code will be returned to the application program in the cmderr variable indicating that an unrecognized document was scanned (error code 507).

SCANNING THE FORM: SCAN

OVERVIEW: GRID

The GRID command is used to resolve an area (or grid) on a form into a string of characters. The GRID command uses the standard NCS mark discrimination techniques to resolve the data in a grid. Refer to Section One for a complete description of how the scanner reads forms and performs mark discrimination.



EXPLANATION: CALL FORMAT

The application program calls the GRID routine and must pass four parameters, one of which is an array of values:

- Gridarg
- Gridstr
- Edstat
- Cmderr

Three parameters are returned to the application program:

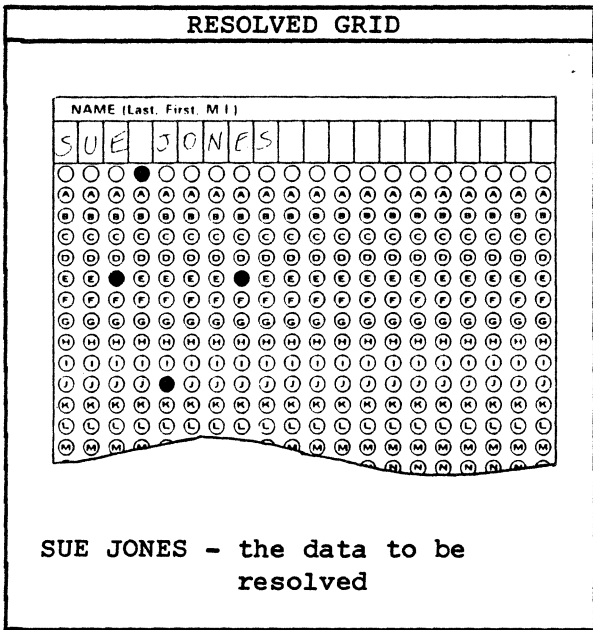
- Gridstr
- Edstat
- Cmderr

CALL FORMAT

```

BASIC:
  FOR A = 0 TO 7
  READ GRIDARG(A):NEXT A
  EDSTAT% = 0
  CMDERR$ = SPACE$(200)
  GRIDSTR$ = SPACE$(200)
  ARGPTR% = VARPTR(GRIDARG%(0))
  CALL GRID%(ARGPTR%,GRIDSTR$,
             EDSTAT%,CMDERR$)
  DATA 2,0,1,10,1,5,5,0

PASCAL:
  VAR
    GRIDARG:GRIDARA;
    EDSTAT:INTEGER;
    CMDERR,GRIDSTR:MSCSTR;
  GRIDARG[0]:= 2;
  GRIDARG[1]:= 0;
  GRIDARG[2]:= 1;
  GRIDARG[3]:= 10;
  GRIDARG[4]:= 1;
  GRIDARG[5]:= 5;
  GRIDARG[6]:= 5;
  GRIDARG[7]:= 0;
  GRID (GRIDARG,GRIDSTR,EDSTAT,
        CMDERR);
    
```



Gridstr

Gridstr is a string variable which returns the result of the GRID operation to the application program.

Edstat

The GRID routine returns error numbers that result from incorrect coding of grids. The programmer must initialize this variable to 0 within the application program before calling the GRID routine. If edstat = 128, an input parameter error has been detected and will be listed in the variable cmderr. If no coding errors or parameter errors occur, the value returned in edstat will be less than 128. The edstat error codes are described in more detail later in this section.

Cmderr

Cmderr is a string variable which returns a parameter error code to the application program from the GRID routine. If no errors occur, cmderr will return the symbol @ in the first character position and the number of characters returned in gridstr. For example if cmderr were @004, no parameter errors were detected and the first four characters of gridstr contained the result of the CALL.

Gridarg

Gridarg is an array of integers which define the GRID to be resolved. They include type, class, sx, ex, ix, sy, ey, and iy.

In Basic, the pointer to the array gridarg is passed to the GRID command routine. The pointer must be set to the starting location of the array. This must be done immediately preceding the GRID call.

Gridarg(0)-Type: Type is a one-digit variable defining the type of grid to be resolved.

GRID TYPES
1. Alphanumeric (space, A-Z, 0-9, special characters)
2. Numeric (0-9)
3. One-Digit Response (1-9)
4. Two-Digit Response (01-99)
5. Binary
6. Binary Coded Decimal (0-9)
7. Litho-Code

When type = 1 the grid type is Alphanumeric (space, A-Z, 0-9, special characters.) An alphanumeric grid can contain up to 63 response positions per item. If it is not necessary to use all 63 spaces, truncation must be made from the end, not the beginning. If the special characters are used, they are placed in this order: [.<(+\&!\$*);^_>?:#@'=" .

GRID TYPE 1 - ALPHANUMERIC

When type = 2 the grid type is Numeric (0-9). The numeric grid may be truncated at the end, but not at the beginning.

GRID TYPE 2 - NUMERIC

SOCIAL SECURITY NUMBER												
0	0	0	0	0	0	0	0	0	0	0	0	0
1	1	1	1	1	1	1	1	1	1	1	1	1
2	2	2	2	2	2	2	2	2	2	2	2	2
3	3	3	3	3	3	3	3	3	3	3	3	3
4	4	4	4	4	4	4	4	4	4	4	4	4
5	5	5	5	5	5	5	5	5	5	5	5	5
6	6	6	6	6	6	6	6	6	6	6	6	6
7	7	7	7	7	7	7	7	7	7	7	7	7
8	8	8	8	8	8	8	8	8	8	8	8	8
9	9	9	9	9	9	9	9	9	9	9	9	9

GRID TYPE 3 - RESPONSE(1-9)

13 14 1 (inch)
 5 (inches)
 4 (inches)
 1-1/4 (inches)
 5/4
 Other
 No Response
 I Don't Know

GRID TYPE 4 - RESPONSE(01-99)

SPORTS INTERESTS
 (Mark only one)

badminton
 baseball
 basketball
 boxing
 cycling
 football
 golf
 gymnastics
 hockey
 skating
 soccer
 swimming
 tennis

GRID TYPE 5 - BINARY

1 2 4 8 16 32 64 128 256

Gridarg(0)-Type (cont.)

When type = 3 the grid type is a one digit Response grid (1-9). Positions in Response grids can be designated with numeric or alpha characters, but the output for both responses is numeric. In both examples, if the third position of the Response grid items are filled in, the output would be a 3. Therefore, it may be necessary for the host program to translate the 1-9 responses into the intended responses. The response grid may be truncated at the end, but not at the beginning.

When type = 4, the grid type is a two-digit Response grid (01-99). Grid response positions can be designated as numeric or alpha characters in any combination, but the output will be in two-digit numeric characters. The grid may be truncated at the end, but not at the beginning.

When type = 5, the grid type is Binary. While the Alphanumeric, Numeric, and Response grids expect only one response per item, Binary grids expect zero or more responses per item. Binary grids also differ from the other grids in that responses do not represent successive numbers (1,2,3,etc.) or letters (a,b,c, etc.), but instead represent successive powers of two (1,2,4,8), etc.). Binary grids may contain a parity bubble for odd parity. The parity bubble must be placed at the high end of the item. Binary grid items

Gridarg(0)-Type (cont.)

can contain up to 29 response positions (or 30 with parity). The maximum number of items in a binary grid is 28. Binary grids cannot generate blanks or multiples. If a response position is left blank, zeros will be generated.

The binary output chart will help you determine how many output characters to count for each item in a binary grid. Note that a parity bubble is not considered as a response position when determining output characters.

The output of a Binary grid item is the numeric total of the values of every bubble marked. In the accompanying example, three binary positions are marked. When each position is added, the total value is 73. (Since there are 9 response positions and numeric values take 3 digits, the returned gridstr will be 073.)

Binary grids are especially useful for machine-coding large numbers in small spaces.

When type = 6 the grid type is Binary Coded Decimal. Like the Binary grid, response positions in a Decimal grid represent values that are powers of two. The values of all positions marked in one item are added together to generate a decimal number in the output record. Decimal grids resolve items of one to five response positions. If five positions are used, the fifth position must be a parity bubble for odd parity.

BINARY OUTPUT CHART		
Number of Responses Positions	Number of Output Characters	Maximum Decimal Value
1 to 3	1	7
4 to 6	2	63
7 to 9	3	511
10 to 13	4	8191
14 to 16	5	65535
17 to 19	6	524287
20 to 23	7	8,388,607
24 to 26	8	67,108,863
27 to 29	9	536,870,911

CODING BINARY GRIDS									
1	2	4	8	16	32	64	128	256	
●	○	○	●	○	○	●	○	○	
$1 + 8 + 64 = 73$									

Gridarg(0)-Type (cont.)

2. Affix a scanner-readable control number to a single page form for distribution or processing control purposes.

Because the printing press has no means of computing when parity bubbles should be marked, resolution of litho code class grids must not have a parity edit. However, no error will be reported if a parity edit is used. Litho code grids contain only one item per grid.

Gridarg(1)-Class

The class variable of gridarg can be 0 through 7. A horizontal grid is class 0. A vertical grid is class 1. If the programmer is using binary grids or binary coded decimal grids, class = 2 or 3 will check to ensure correct parity and return the check in the edit variable. Classes 4 through 7 are the same as classes 0 through 3 except that they are linked grids.


CLASS				
Value	Horiz	Vert	Parity	Link
0	X			
1		X		
2	X		X	
3		X	X	
4	X			X
5		X		X
6	X		X	X
7		X	X	X

A grid is vertical if response positions run parallel to the timing track and horizontal if response positions run perpendicular to the timing track.

VERTICAL				
BIRTH DATE				
	Yr.	Mo.		
■	0	0	0	0
■	1	1	1	1
■	2	2	2	2
■	3	3	3	3
■	4	4	4	4
■	5	5	5	5
■	6	6	6	6
■	7	7	7	7
■	8	8	8	8
■	9	9	9	9

HORIZONTAL					
	1	2	3	4	5
6	⊗	⊗	⊗	⊗	⊗
	1	2	3	4	5
7	⊗	⊗	⊗	⊗	⊗
	1	2	3	4	5
8	⊗	⊗	⊗	⊗	⊗
	1	2	3	4	5
9	⊗	⊗	⊗	⊗	⊗
	1	2	3	4	5
10	⊗	⊗	⊗	⊗	⊗

Gridarg(1)-Class (cont.)

LINKING EXAMPLES					
1. -a b c d e					
-o o o o o					
	segment	segment	segment.		
	1	2	3		
2.-o	1885	o 1906	o 1923		
-					
-o	1896	o 1915	o 1931		
-					
-o	1897	o 1918			
-					
-o	1899	o 1919			
					
single item					

Linking: Not all items are set in uniform grids. Occasionally one question may consist of several non-uniform sets of responses, such as the grid in example one.

Each of the uniform sets of responses is resolved with a separate grid call. However, since just one output is desired for the item, there must be a way of connecting the resolutions. Linking connects the contents of two or more item segments into one output. It may be used with any grid type, but is primarily used to link response grids. Grids which use linking can contain only one item.

Example two is a single item composed of three distinct segments (in this case, response strings.) Each response string is resolved with an individual grid call. Grid call one would resolve 1885-1899. Grid call two would resolve 1906-1919; and grid call three would resolve 1923-1931.

To link these strings together, a special grid class must be used. Grid classes 4 through 7 are really the same as classes 0 through 3 except that the linking provision is added.

Linking (cont.)

To link one segment to the next, use the appropriate grid class (with linking) in each grid call until reaching the last segment. Then make the grid call using the appropriate class without linking. This signals the program that the last segment in the item has been reached. At this point the grid routine will return the selected response in the grid-string variable.

In the second grid example the first grid class would be linked vertical (class=5). This links the first segment to the next. The second grid class would also be linked vertical (class=5). This links the first and second segments to the next. The third grid class would be vertical with no linking (class=1). This signals that there is no more linking and the response (01-10) is returned by the grid routine in the variable gridstring.

Note: The same restrictions which apply to all other grids also apply to linked grids. Linked item segments must be of one type and item lengths cannot exceed normal length limits. For example, a linked two-digit response grid (01-99) cannot have more than 99 response positions.

Gridarg(2)-Sx: The variable sx (start x) indicates the starting x coordinate of the grid. To locate this position, the programmer must use an NCS Sheet Compile Ruler. Then find sx by following this procedure.

1. Lay the ruler parallel to the leading edge of the form (the edge with the skunk marks) using 6-to-the-inch scale.

LINKING EXAMPLES (cont.)

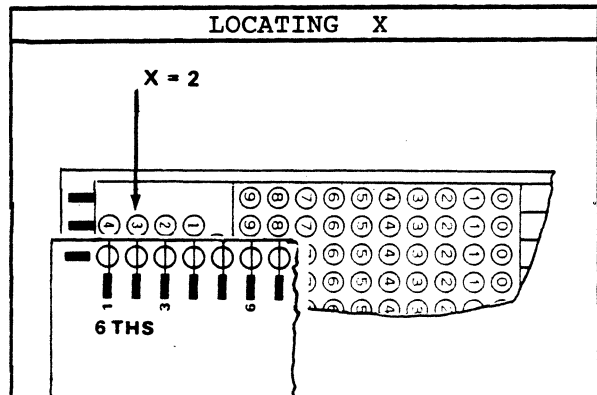
GRID CALLS - EXAMPLE 2

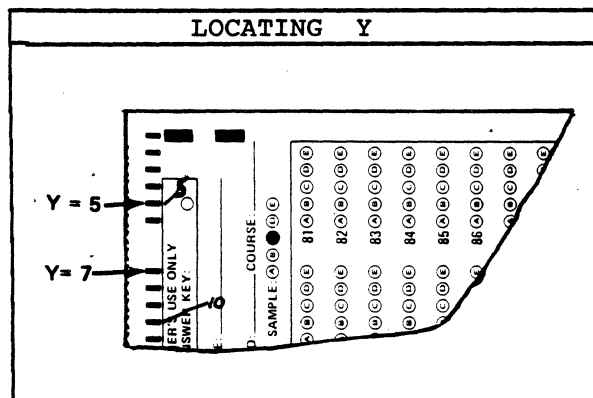
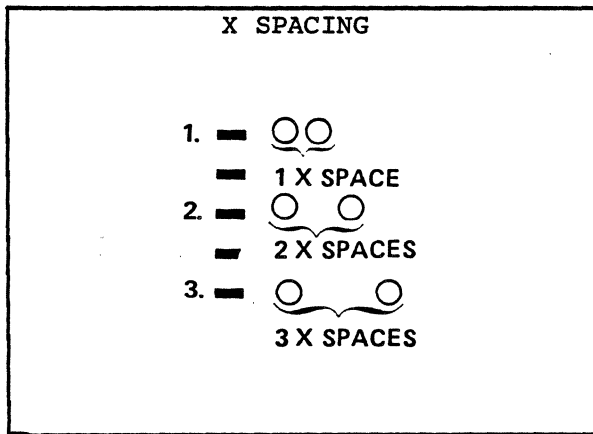
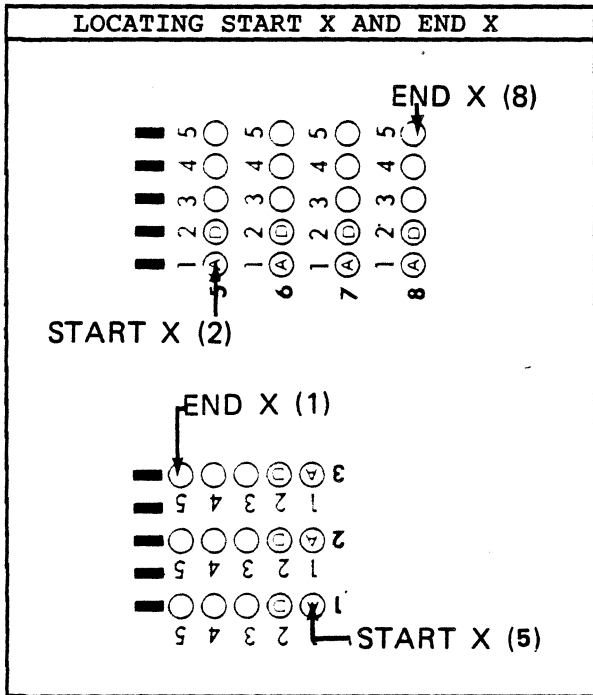
```
string 1 CALL GRID (...class =
          5- linked vertical)

string 2 CALL GRID (...class =
          5- linked vertical)

string 3 CALL GRID (...class =
          1- vertical)
```

LOCATING X





Gridarg(2)-Sx (cont.)

2. Line up the timing mark on the ruler with the timing mark on the form.
3. The distance in x units from the timing track to the first response choice of the first item is sx (Start x).

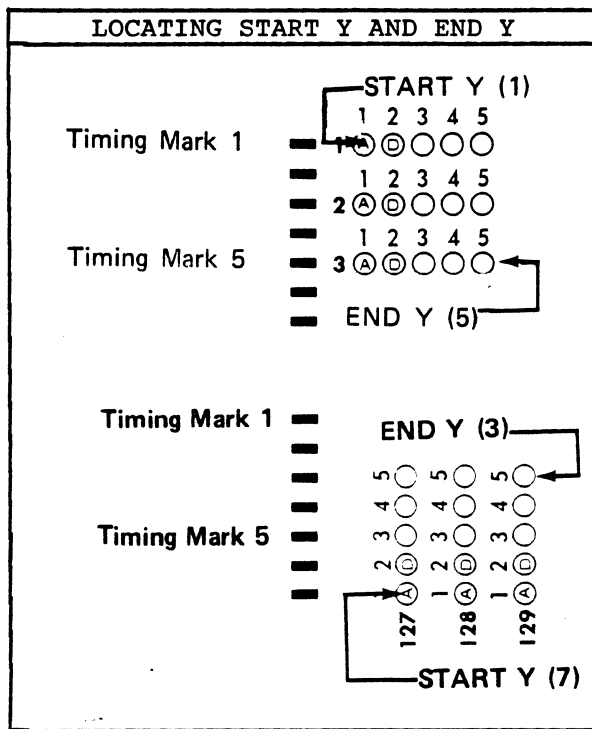
Gridarg(3)-Ex: The variable ex (end x) indicates the ending x coordinate of the grid. It is found through the same procedure described in the sx (start x) section, except the distance in x units is to the last response of the last item.

Gridarg(4)-Ix: The variable ix (increment x) indicates the spacing between x response items. X-axis spacing is fixed at six response positions per inch. An NCS Sheet Compile Ruler is used to measure these spaces. In example one, the x spacing is one since there are no spaces in between x response items. In example two, the x spacing is two since there is one space in between x responses. In example three, the x spacing is three since there are two spaces in between x responses.

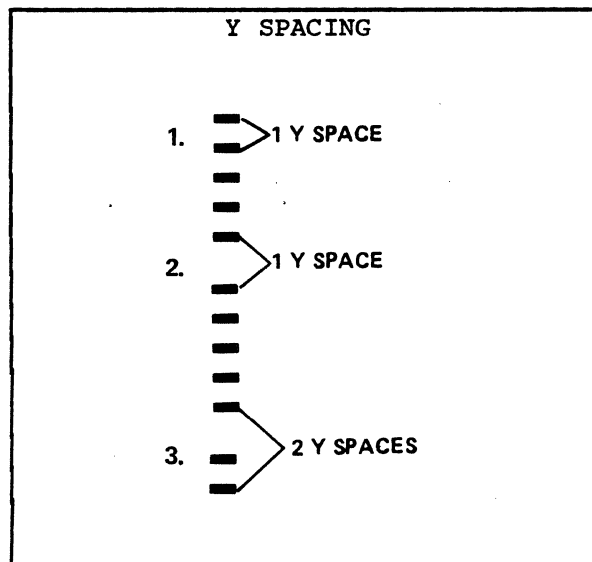
Gridarg(5)-Sy: The variable sy (start y) indicates the starting y coordinate of the grid. To locate this position, the programmer must:

1. Mark every fifth timing mark for reference.
2. Count down from the closest marked timing mark to the timing mark corresponding to the first response position for sy (start y).

Gridarg(6)-Ey: The variable ey (end y) indicates the ending y coordinate of the grid. It is found through the same procedure described in the sy (start y) section, except the count is to the timing mark corresponding to the last response position for ey (end y).



Gridarg(7)-Iy: The variable iy (increment y) indicates the spacing between y response items. In examples one and two, the y spacing is one since there is one space between y response items. In example three, there are two spaces between y response items.



EDSTAT RETURNS:		
VALUE	BIT	GRID CONDITION
1	0	at least 1 omit
2	1	at least 1 multiple
4	2	incomplete/multiple
8	3	blank grid
16	4	not left justified
32	5	not right justified
64	6	parity error (binary & binary coded decimal)
128	7	call had parameter error

OMIT	
1. ○ ● ○ ○	4. ● ○ ○ ○
2. ○ ○ ○ ●	5. ○ ○ ○ ○
3. ○ ○ ○ ●	6. ○ ○ ○ ●

MULTIPLE RESPONSE	
1. ○ ● ● ○	

INCOMPLETE/MULTIPLE	
1. ● ○ ○ ○	5. ○ ○ ● ○
2. ○ ○ ○ ○	6. ○ ● ○ ○
3. ○ ○ ○ ○	7. ○ ○ ○ ●
4. ○ ○ ○ ○	8. ○ ● ● ○

BLANK GRID	
1. ○ ○ ○ ○	4. ○ ○ ○ ○
2. ○ ○ ○ ○	5. ○ ○ ○ ○
3. ○ ○ ○ ○	6. ○ ○ ○ ○

Edstat

The GRID routine returns the variable edstat -- a decimal variable which can be divided into 7 bits.

Edstat reports whether a grid has been correctly filled out. Each bit describes a specific edit condition. If the bit contains zero, the grid condition for that location is not present. If the bit contains one, the grid condition is present.

Bits 0-7 indicate special grid conditions.

A one in bit 0 indicates that at least one item has been omitted. The string of characters in the variable gridstr will contain a blank at the omitted location.

A one in bit 1 indicates a multiple response, meaning that more than one response has been chosen in at least one item. The data string in gridstr will contain an asterisk (*) at the multiple location.

A one in bit 2 signals that the grid has been left incomplete or that a multiple response has occurred. Check bits 0 and 1 to see which situation is present.

A one in bit 3 signals that the grid has been left entirely blank.

INTERPRETING EDSTAT

EDSTAT = 21

128	64	32	16	8	4	2	1	value
0	0	0	1	0	1	0	1	0 or 1
7	6	5	4	3	2	1	0	position

21	
-16	←
5	
-4	←
1	
-1	←
0	

EXAMPLE: INTERPRETING EDSTAT

Since edstat is returned as a decimal number, the user must be able to divide the number into its binary equivalent to interpret the grid edit condition. Each bit position (0-7) has a value equal to a power of two. By taking the edstat value and subtracting each greatest possible power of two until reaching zero, the user can determine which bits contain a one. The example to the right accomplishes that task. If the number returned in edstat is 21, the first value which is not greater than 21 is 16 (position 4). $21 - 16 = 5$. The next value not greater than 5 is 4 (position 2). $5 - 4 = 1$, and $1 - 1 = 0$ (position 0).

GRID CONDITION

IDENTIFICATION NUMBER									
	1	2	7	3	0	2	5	9	

0	0	0	0	0	0	0	0	0	0
1	●	1	1	1	1	1	1	1	1
2	2	●	2	2	2	●	2	2	2
3	3	3	3	●	3	3	3	3	3
4	4	4	4	4	4	4	4	4	4
5	5	5	5	5	5	5	●	5	5
6	6	6	6	6	6	6	6	6	6
7	7	7	●	7	7	7	7	7	7
8	8	8	8	8	8	8	8	8	8
9	9	9	9	9	9	9	9	●	9

incomplete, with omit on left

Therefore, positions 0, 2 and 4 contain ones. The grid condition is incomplete (position 2) with an omit (position 0) occurring on the far left (position 4).

EDSTAT - BINARY GRID

POSITION	CONTAINS
0	always 0
1	always 0
2	1 when parity error
3	always 0
4	always 0
5	always 0
6	1 when parity error
7	1 when parameter error

EDSTAT: BINARY GRID

The variable edstat is interpreted differently for binary grids. Positions 0,1,3,4, and 5 remain constant while bits 2, and 6 signal parity errors and bit 7 signals an error in passing parameters.

EDSTAT: BINARY CODED DECIMAL GRID

The variable edstat is interpreted differently for binary coded decimal grids. Bits 0,3,4, and 5 remain constant while bit 1 signals that an item is coded > 9, bits 2 and 6 signal parity errors and bit 7 signals an error in passing parameters.

CONSIDERATION: EDSTAT, CMDERR

The variable edstat contains errors that result from incorrectly coding the grids on forms. In the absence of parameter errors, cmderr returns the @ symbol. However, if edstat = 128, at least one parameter error has been made and cmderr contains the parameter error number(s). Programming error numbers are described in Section Six.

EXPLANATION: SAMPLE PROGRAMMING

This program section scans 25 forms and resolves a name grid from each form to compile a list of class members. The program section assumes that the parameters for the SCAN call have been defined previously. Line 20 prints a heading for the class list. Lines 30 through 140 comprise a loop which scans forms and resolves name grids. The loop runs 25 times, once for each student. Line 40 calls the SCAN command and a form is scanned. Lines 50 and 60 initialize EDSTAT% and GRIDSTR\$. Lines 80 and 150 read in parameters for the GRID call through READ and DATA statements. Line 90 initializes CMDERR\$. Line 100 sets the pointer to the start of the GRIDARG array.

EDSTAT - BINARY CODED DECIMAL

BIT	CONTAINS
0	always 0
1	1 when coded # > 9
2	1 when parity error
3	always 0
4	always 0
5	always 0
6	1 when parity error
7	1 when parameter error

EDSTAT/CMDERR

if no parameter errors:

```
edstat=decimal number (0-127),
      numbers 1-127 indicate
      grid edit conditions
cmderr=@
```

if parameter errors:

```
edstat=128 (indicating parameter
           error(s))
cmderr=parameter error number
      string
```

SAMPLE PROGRAMMING

```
20 PRINT "CLASS MEMBERS ARE:"
30 FOR B= 1 TO 25
40 CALL SCAN%(DOC%,READTYPE%,
              CMDERR$)
50 EDSTAT% = 0
60 GRIDSTR$ = SPACE$(200)
70 FOR A = 0 TO 7
80 READ GRIDARG%(A):NEXT A
90 CMDERR$ = SPACE$(200)
100 ARGPTR% = VARPTR(GRIDARG%(0))
110 CALL GRID%(ARGPTR%,GRIDSTR$,
              EDSTAT%,CMDERR$)
120 IF EDSTAT% = 128 THEN GOTO 200
130 PRINT GRIDSTR$
140 NEXT B
150 DATA 1,0,10,28,1,5,27,1
200 error check section...
```

RESOLVING THE GRIDS: GRID

EXPLANATION: SAMPLE PROGRAMMING

(cont.)

Line 110 calls the GRID routine which resolves the name grid. If EDSTAT% = 128 (line 120), a parameter error has occurred and the program is directed to line 200 which is the start of the program's parameter error check. If EDSTAT% is not 128, no parameter error has occurred and the program prints GRIDSTR\$ which contains the name from the resolved grid (line 130). Line 140 directs the program back to line 30 and the sequence is repeated until all 25 names are read.

OVERVIEW: TRANSMIT

The TRANSMIT command allows data to be transmitted to the following destinations: the LED display, the optional transport printer, or the auxiliary port.

EXPLANATION: CALL FORMAT

The application program calls the TRANSMIT routine and must pass four parameters:

- Dest
- Outstr
- Prntpos
- Cmderr

One parameter is passed back to the application program:

- Cmderr

Dest

Dest (destination) is a one-digit variable which indicates to which device output is to be directed. Dest will be 0 for the scanner transport printer, 1 for the scanner auxiliary port, and 2 for the scanner LED display. The length of the data transmitted is dependent on the device being selected. The transport printer is limited to 99 characters. There is a 254 character limit to the auxiliary port and the LED display is limited to one numeric character.

CALL FORMAT	
BASIC:	
DEST% = 0	
CMDERR\$ = SPACE\$(200)	
PRNTPOS% = 10	
OUTSTR% = SPACE\$(200)	
CALL TRANSMIT%(DEST%,OUTSTR\$, PRNTPOS%,CMDERR\$)	
PASCAL:	
VAR	
CMDERR,OUTSTR:MSCSTR;	
DEST,PRNTPOS:INTEGER;	
DEST:= 0;	
PRNTPOS:= 10;	
OUTSTR:= "ERROR";	
TRANSMIT(DEST,OUTSTR,PRNTPOS, CMDERR);	

DEST		
dest number	which device	length of output or prntpos
0	transport printer	99
1	aux. port	254
2	LED display	1

SENDING DATA TO SCANNER: TRANSMIT

Cmderr

Cmderr is a string variable which returns an error code to the application program from the TRANSMIT routine.

PRNTPOS	
if DEST =	PRNTPOS is
0	01 to 99
1	0
2	1 to 15

Prntpos

Prntpos is the name of a number variable. If the destination is the transport printer (DEST = 0) then prntpos will be the desired starting print position (from 01 to 99). If the destination is the auxiliary port (DEST = 1) then prntpos will be zero. If the destination is the scanner display (DEST = 2) then prntpos will be the value the programmer wants displayed (0-15). Numbers 10-15 appear as A,b,C,d,E, and F.

Outstr

Outstr is a string variable containing the data to be transmitted. First, the user must decide which data to transmit and then load that data into outstr.

EXPLANATION: SAMPLE PROGRAMMING

This program section utilizes both the TRANSMIT and RECV commands. A test grading center has been set up with the microcomputer operator at the host computer and an instructor at a remote location with a display monitor and keyboard attached to the scanner's auxiliary port. This program section describes the start of the test grading procedure. The host sends a message to the auxiliary port asking if the scanner operator is ready to begin scanning forms. The host expects a response from the scanner operator.

Lines 40-60 initialize the TRANSMIT parameters. The message which will be sent to the aux port is entered in OUTSTR\$. Line 80 makes the TRANSMIT CALL and the message is sent. In line 90, if CMDERR\$ does not contain "@" (ASCII 64) the call has not been made correctly and the program is directed to line 200, the start of the TRANSMIT error check.

If no errors occurred during the TRANSMIT call, line 100 initializes RECVSTR\$. Then the RECV CALL is made and the "Y" or "N" the scanner operator entered on the aux port keyboard is received by the host (line 120). In line 130, if CMDERR\$ does not contain "@" (ASCII 64), an error has been made in the call and the RECV error check is started in line 300. If "Y" is returned in RECVSTR\$, the scanner operator is ready to begin scanning forms and the scanning routine begins (line 400). If "N" is returned in RECVSTR\$, the scanner operator is not ready to begin scanning forms and the program moves to line 100 which waits for a response of "Y" from the scanner operator.

SAMPLE PROGRAMMING

```

40 CMDERR$ = SPACE$(200)
50 DEST% = 1
60 PRNTPOS% = 0
70 OUTSTR$ ="ARE YOU READY?(Y/N)"
80 CALL TRANSMIT(DEST%,PRNTPOS%,
                OUTSTR$,CMDERR$)
90 IF ASC(CMDERR$)<>64 THEN GOTO
   200
100 RECVSTR$ = SPACE$(80)
110 CMDERR$ = SPACE$(200)
120 CALL RECV(RECVSTR$,CMDERR$)
130 IF ASC(CMDERR$)<>64 THEN GOTO
   300
130 IF RECVSTR$="Y"THEN GOTO 400
140 IF RECVSTR$<>"N"THEN GOTO 100
200 TRANSMIT error section...
300 RECV error section...
400 testing routine...

```

SENDING DATA TO SCANNER: TRANSMIT

OVERVIEW: RECV

The RECV command allows the host to receive a data string of up to 254 characters from an auxiliary device. The command is especially useful when the user has reason to enter data via an auxiliary keyboard rather than the scanner. For example, if an instructor were scanning batches of test forms for three different classes and certain students did not complete the class grid on the test form, the instructor could enter that data on the keyboard.

EXPLANATION: CALL FORMAT

The application program calls the RECV routine and must pass two parameters:

- Recvstr
- Cmderr

Two parameters are passed back to the application program:

- Recvstr
- Cmderr

Recvstr

Recvstr is the name of a string variable which is received from the auxiliary device. The operator enters the string and presses the return key on the auxiliary device to terminate the string. Up to 255 characters can be received by the RECV CALL in Basic. In Pascal, the limit is 254 characters.

Cmderr

Cmderr is a string variable which returns an error code to the application program from the RECV routine.

CALL FORMAT

```

BASIC:
  RECVSTR$ = SPACE$(255)
  CMDERR$ = SPACE$(200)
  CALL RECV$(RECVSTR$,CMDERR$)

PASCAL:
  VAR
    RECVSTR,CMDERR:MSCSTR;
  RECV (RECVSTR,CMDERR);

```

RECEIVING AUX DEVICE DATA: RECV

EXPLANATION: SAMPLE PROGRAMMING

A program section which illustrates both the TRANSMIT and RECV command appears in this section under Sending Data to the Scanner, page 3-45.

4

HOST
PROGRAM

Introduction.....4-2
Program Structure.....4-3

INTRODUCTION

Using the scanner as an input device necessitates changes in application programming. Although Scanner Commands simplify many procedures, certain programming steps must be initiated to allow the programmer to use scanner commands. This section describes the structure of an application program that uses Scanner Commands.

OVERVIEW: PROGRAM STRUCTURE

In general, all programs must accomplish three things. First, a means must be provided for data input. Second, the data must be manipulated to achieve a desired result. Third, the result must be presented (or output) in a meaningful manner.

A program which uses data input from scannable forms must accomplish certain other tasks including establishing buffers and loading commands into memory.

This section describes the general format of an application program which utilizes scanner commands.

EXPLANATION: LOADING OR ACCESSING SCANNER COMMANDS

To utilize Micro Scanner Commands, the programmer must insert special statements within the application program. These statements tell the microcomputer to load scanner commands in memory. The scanner commands then exist in a portion of memory which is protected from inadvertent overwriting.

Interpretive Basic

The first line of the Basic program should define the beginning segment of memory that the commands will be loaded into. This location should be set to &H1CD4. The following lines load the commands into the location defined in the first line.

PROGRAM SEQUENCE

Load or access commands

LOADING SCANNER COMMANDS
FOR INTERPRETIVE BASIC

```
DEF SEG =&H1CD4
BLOAD "COMNDS.BAS",0
SCAN%=PEEK(0) + 256*PEEK(1)
GRID%=PEEK(2) + 256*PEEK(3)
SKUNK%=PEEK(4) + 256*PEEK(5)
CONTROL%=PEEK(6) + 256*PEEK(7)
SETUP%=PEEK(8) + 256*PEEK(9)
LEVEL%=PEEK(10) + 256*PEEK(11)
TRANSMIT%=PEEK(12) + 256*PEEK(13)
RECV%=PEEK(14) + 256*PEEK(15)
```

PROGRAM STRUCTURE

ACCESSING SCANNER COMMANDS FOR PASCAL

```
(*$INCLUDE:'SCANDECL.PAS'*)
```

Pascal

The commands can be easily accessed by including one statement "(*\$INCLUDE:'SCANDECL.PAS'*)". This statement should immediately follow the program heading line.

Compiled Basic

The commands are linked to the host program and no special statements are required in the host program.

EXPLANATION: DECLARING VARIABLES

Pascal

In Pascal, variable types must be declared and input before they are passed to command routines. Most command routine variables are of type INTEGER. However, the cmderr variable, which passes back error messages or resolved grid data to command routines, is a special MSCSTR variable. The variable marks, the array of skunk mark locations, is type SKARAY. The variables recvstr and outstr, which transport data to and from the host computer, are of type MSCSTR. Gridarg, the array containing parameters which describe a grid, is type GRIDARA. It is not necessary to declare types MSCSTR, SKARAY, or GRIDARA within the type declaration since these types are already declared within the SCANDECL.PAS file which is included in the host program.

Basic

No declaration section is necessary in Basic since the variable type is implicitly declared within the variable name. All variables ending with "%" are type integer. Those ending with "\$" are string variables.

PASCAL - VARIABLE TYPES

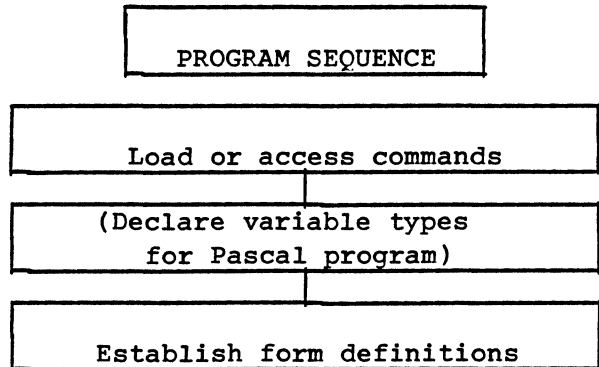
<u>Variable</u>	<u>Command</u>	<u>Type</u>
Cmderr	all	MSCSTR
Marks	SKUNK	SKARAY
Recvstr	RECV	MSCSTR
Outstr	TRANSMIT	MSCSTR
Gridarg	GRID	GRIDARA
All others	----	INTEGER

Where:

```
MSCSTR = LSTRING (254);  
SKARAY = ARRAY[0..47] of integer;  
GRIDARA = ARRAY[0..7] of integer;
```

EXPLANATION: FORM DEFINITIONS

After variables have been declared, the forms to be used in the program should be defined with the SKUNK command. All form definitions should be listed in this area. This makes the program more readable since all definitions are in one location. It also saves programming time since the programmer does not have to consider form definitions throughout the program.



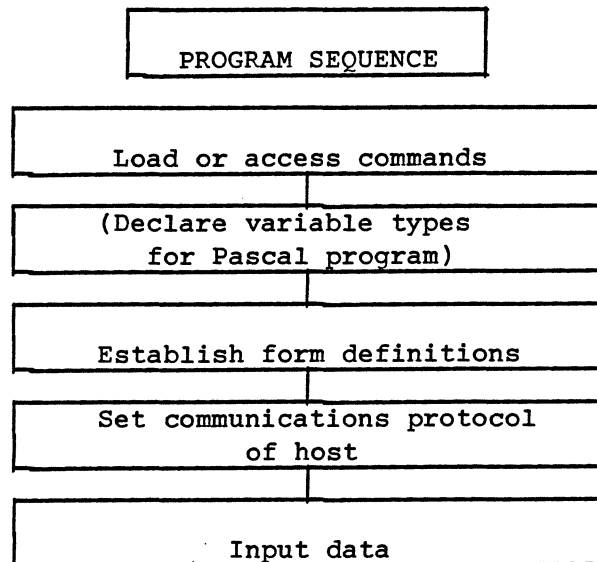
EXPLANATION: COMMUNICATIONS PROTOCOL

Before obtaining data from the scanner or auxiliary device, the communications protocol should be checked to ensure that the host and scanner (or auxiliary device) can communicate successfully. The host protocol must be configured to match that of the scanner or auxiliary device. The protocol can be programmed into the host through the SETUP command.

EXPLANATION: INPUT DATA

The next step in an application program is to input the data that a program will work with.

In Basic, variables must be initialized (that is, set to a value) before they are sent to the command routines. Parameters which are not used as input to the command routines, but which return values to the host program, also need to be initialized at this point. In Basic, the variable cmderr, which returns error messages, should be set to SPACE\$(200). Recvstr, the variable containing data sent to the host by the auxiliary device in the RECV command, should be set to SPACE\$(255).



In Pascal, only variables which pass values to the command routines need to be set prior to the call.

EXPLANATION: INPUT DATA (cont.)

Data can be input in a variety of ways. It can be entered by scanning forms (using the SCAN command). It can be entered through an auxiliary device (using the RECV command). Or if data is pre-determined (like heading information), it can be entered from within the program. (In Basic this would be accomplished with READ and DATA statements.)

CONSIDERATION: INPUT DATA

It is possible, during the course of the SCAN, TRANSMIT, or RECV commands, that the scanner will not transfer control back to the program. This could happen in the case of a communications error or when a sheet jams and the scanner doesn't send an EOR (end of record) signal. The operator can abort the routine by pressing the 'ESC' (escape) key on the host keyboard.

EXPLANATION: RESOLVE DATA

Once the SCAN command transfers data from a form to the host, the data must be resolved. That is, data for each grid must be taken from the sheet buffer and transferred from read levels to data for use in the application program. This is a simple process, since the GRID command resolves grid data one grid at a time.

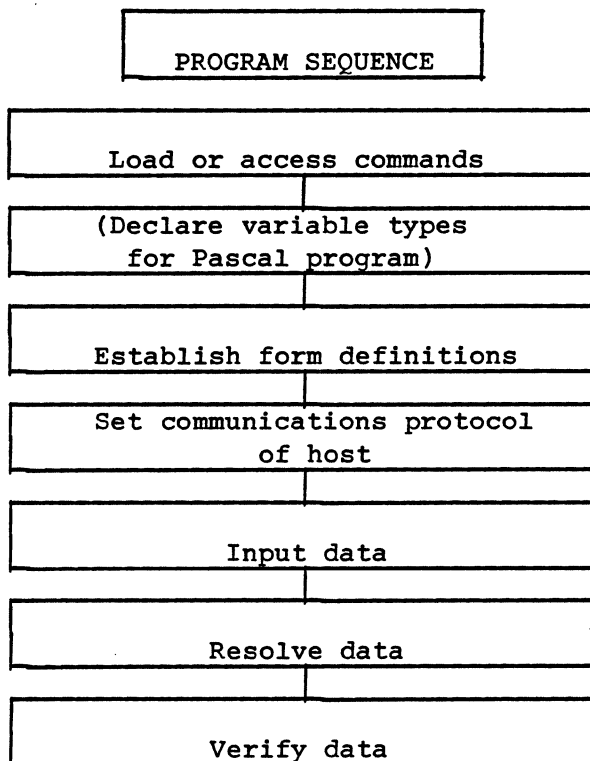
EXPLANATION: VERIFY DATA

The application program must check the validity of data input during the program. Communications errors can occur, in which case the program should request retransmission of data. (If this happens while entering data from the scanner, use the SCAN command with option 3, retransmit data.) Or if the entered data is not what the program expects, an error message should be displayed with suggestions of how to correct the error.

Parameter errors or errors due to filling out a form incorrectly can be determined by interpreting the variable edstat. For a complete description of the edstat variable, refer to Section Three, Resolving the Grids.

For example, a program is designed to record pertinent data (such as company division, employee address, health insurance number) for all employees in a company. Since employees are referenced by social security number, it is essential that the social security grid be complete. The program section uses the GRID command to resolve the social security grid and checks for errors in grid completion.

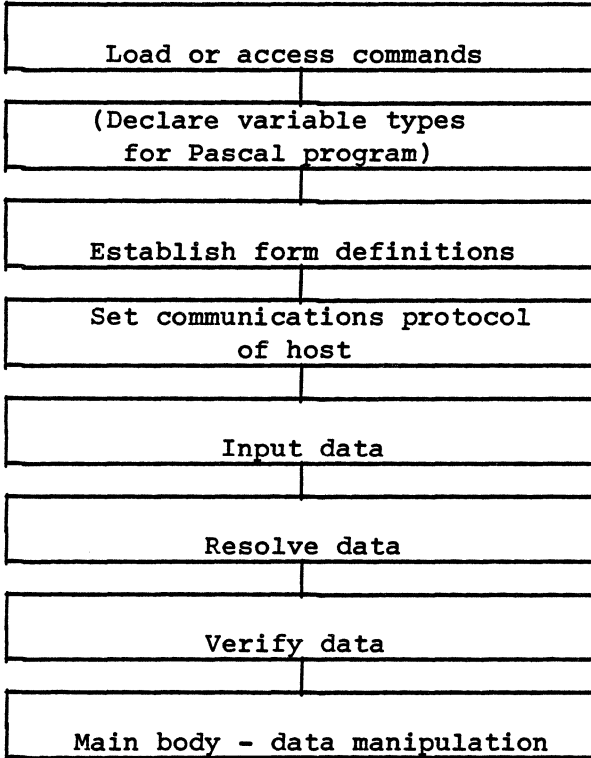
First the GRID call is made. Then, if the edstat variable is less than 64, an error has been made in the completion of the grid. (For an interpretation of edstat, refer to Section Three, Resolving the Grids.) An operator message is provided in line 220 telling the operator to correct the social security grid and rescan the sheet.



VERIFY DATA	
210	CALL GRID%(ARGPTR%,GRIDSTR\$, EDSTAT%,CMDERR\$)
220	IF EDSTAT% < 64 THEN PRINT "SOCIAL SECURITY GRID COMPLETION ERROR, CORRECT ERROR, RESCAN SHEET" ELSE...

PROGRAM STRUCTURE

PROGRAM SEQUENCE



EXPLANATION: DATA MANIPULATION

The manipulating of data to achieve the desired program result is the main body of the program. In both Basic and Pascal, programmers can use loops or subroutines to achieve results.

RESPONSE ITEM

Which sport do you enjoy most?

- tennis
- basketball
- football
- baseball
- track

EXPLANATION: TRANSLATION OF RESPONSE GRIDS

Although the data returned in alpha and numeric grids can be used without making a translation, data from response grids must often be translated. For example, the response item to the left would yield a 1, 2, 3, 4 or 5. Since the response is returned as 5 by the GRID routine, it must be translated into "track."

EXPLANATION: OUTPUT DATA

After manipulations are complete, results must be output in some manner. Data can be printed on forms (if your scanner is equipped with a transport printer), on a printer attached to the microcomputer, or on the microcomputer screen, etc. It is important that output be meaningful. Output is meaningful if headings are included, lists, or comparisons made. The range of output style is unlimited and greatly enhances the substance and clarity of a program.

CONSIDERATION: OPERATOR INSTRUCTIONS

One sign of a well written program is the abundance of operator instructions. Since the programmer is not always aware of how knowledgeable the user is about the system, operator instructions must be instructive and complete. The programmer must be especially careful when writing instructions regarding errors. For example, an error takes place due to unacceptable data. The data entered is negative and should not be. The following message would not be descriptive:

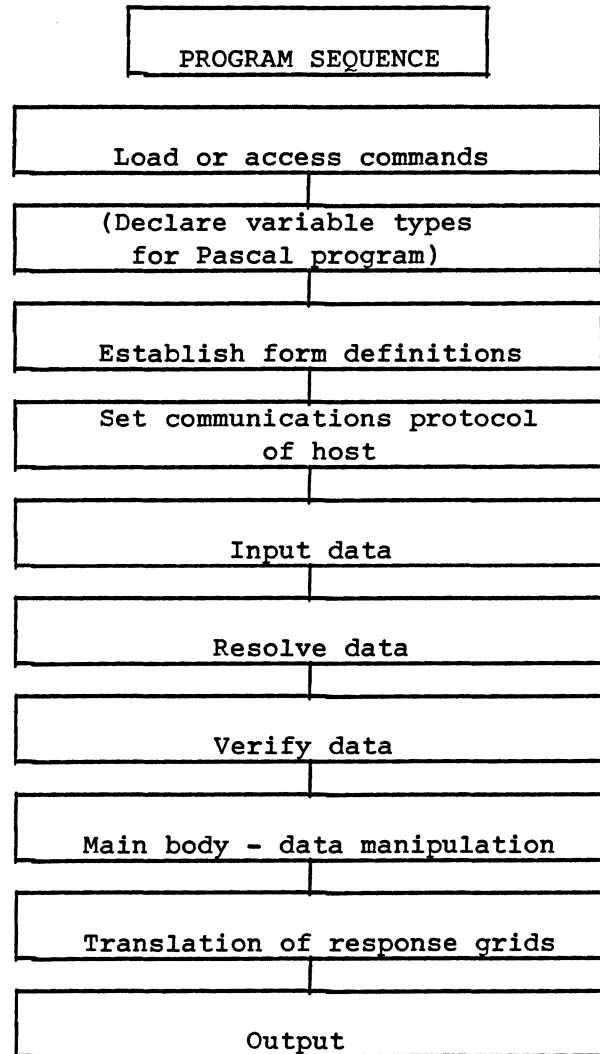
ERROR NOTED

A more meaningful message would be:

ERROR, NEGATIVE DATA NOT ALLOWED

An even more descriptive error message would explain what error took place and how to correct the error:

ERROR, NEGATIVE DATA NOT ALLOWED
REMOVE SHEET FROM SCANNER
RE-MARK SHEET AND RESCAN



PROGRAM STRUCTURE

5

SAMPLE
PROGRAM

Introduction.....5-2
Program Listing (Compiled
Basic).....5-3
Program Explanation.....5-4
Interpretive Basic Listing...5-9

INTRODUCTION

NCS Micro Scanner Commands has been designed to allow users of microcomputers to incorporate input data into programs from scannable forms. The following sample program has been written to give programmers suggestions on how to utilize scanner commands in application programming.

OVERVIEW: PROGRAM FUNCTION

The sample program is designed to scan and report results of the Computest Answer Sheet. The program makes use of the SKUNK, SCAN, and GRID commands, which are generally used together. It also uses the SETUP command to establish communications protocol. The program is coded in Compiled Basic and is accompanied by a line by line explanation. Then the program is listed in Interpretive Basic.

EXPLANATION: PROGRAM

Before its explanation, the Compiled Basic program is listed in one spot, for convenience.

COMPUTEST ANSWER SHEET

Computest

USE NO. 2 PENCIL ONLY

	IDENTIFICATION NUMBER	
	1	()
	2	()
	3	()
	4	()
	5	()
	6	()
	7	()
	8	()
	9	()
	10	()
	11	()
	12	()
	13	()
	14	()
	15	()
	16	()
	17	()
	18	()

	DO	
	NOT	()
	WRITE	()
	IN	()
	THIS	()
	AREA	()
	19	()
	20	()
	21	()
	22	()
	23	()
	24	()
	25	()
	26	()
	27	()
	28	()
	29	()
	30	()
	31	()
	32	()
	33	()
	34	()
	35	()
	36	()

	DO	
	NOT	()
	WRITE	()
	IN	()
	THIS	()
	AREA	()
	37	()
	38	()
	39	()
	40	()
	41	()
	42	()
	43	()
	44	()
	45	()
	46	()
	47	()
	48	()
	49	()
	50	()
	51	()
	52	()
	53	()
	54	()
	55	()

PROGRAM LISTING-COMPILED BASIC

]LIST

```

10 DEFINT A-Z
20 DIM MARKS (2),GRIDARG (7)
30 OPEN "SCANREC" FOR OUTPUT AS #1
40 CLS: PRINT TAB (20) "SAMPLE
PROGRAM FOR NCS/IBM SCANNER
COMMANDS"
50 REM*****
60 CTRLPT=4
70 CMDERR$=SPACE$(20)
80 CALL CONTROL (CTRLPT,CMDERR$)
90 IF LEFT$(CMDERR$,1)<>"@" THEN
PRINT "CONTROL ERROR - ";
CMDERR$:STOP
100 REM*****
110 BAUD =9600
120 PARITY = ASC("O")
130 DATABITS = 7
140 STOPBITS = 2
150 PORTSEL = 1
160 CMDERR$ = SPACE$(20)
170 CALL SETUP (BAUD,PARITY,DATABITS,
STOPBITS,PORTSEL,CMDERR$)
180 IF LEFT$(CMDERR$,1)<>"@" THEN
PRINT "SETUP ERROR - ";
CMDERR$:STOP
190 REM*****
200 OFFSET = -1
210 CMDERR$ = SPACE$(20)
220 CALL LEVEL (OFFSET,CMDERR$)
230 IF LEFT$(CMDERR$,1)<>"@" THEN
PRINT "LEVEL ERROR -";
CMDERR$:STOP
240 REM*****
250 DOCNUM = 1
260 CELLS = 18
270 TRACKS = 39
280 NUMMARKS = 2
290 MARKS (0) = 1
300 MARKS (1) = 7
310 CMDERR$ = SPACE$(20)
320 ARGPTR = VARPTR(MARKS(0))
330 CALL SKUNK (DOCNUM,CELLS,TRACKS,
NUMMARKS,ARGPTR,CMDERR$)

```

```

340 IF LEFT$(CMDERR$,1)<>"@" THEN
PRINT "SKUNK ERROR -";
CMDERR$:STOP
350 REM*****
360 LOCATE 20,20:PRINT "Feed sheet
into scanner or press ESC";
370 DOC = 0
380 READTYPE = 2
390 CMDERR$ = SPACE$(20)
400 CALL SCAN (DOC,READTYPE,CMDERR$)
410 IF LEFT$(CMDERR$,3) ="504" THEN
PRINT PROGRAM ENDED":CLOSE:END
420 IF LEFT$(CMDERR$,1)<>"@" THEN
PRINT "SCAN ERROR -";
CMDERR$:STOP
430 REM*****
440 RESTORE 610
450 A$=""
460 FOR I=1 TO 8
470 FOR J=0 TO 7
480 READ GRIDARG(J)
490 NEXT J
500 GRIDSTR$ = SPACE$(20)
510 CMDERR$ = SPACE$(20)
520 EDSTAT = 0
530 ARGPTR = VARPTR(GRIDARG(0))
540 CALL GRID(ARGPTR, GRIDSTR$,
EDSTAT,CMDERR$)
550 IF LEFT$(CMDERR$,1)<>"@" THEN
PRINT "GRID ERROR -";
CMDERR$:STOP
560 A$=A$+GRIDSTR$
570 NEXT I
580 PRINT #1,A$
590 LOCATE 5,1:PRINT A$
600 GOTO 350
610 DATA 2,0,10,1,1,3,12,1
620 DATA 3,1,18,1,1,15,19,1
630 DATA 3,1,18,1,1,25,29,1
640 DATA 3,1,18,1,1,35,39,1
650 DATA 3,1,18,1,1,34,30,1
670 DATA 3,1,18,14,1,14,10,1
680 DATA 3,1,18,14,1,7,3,1

```

OVERVIEW: PROGRAM LAYOUT

The sample program is described in this section. Program lines are listed on one side of each page with an explanation of each program line on the opposite page. The program is coded in Compiled Basic. Following the program explanation, the same program is coded in Interpretive Basic and Pascal.

Lines 10 through 30 are declarative lines. Line 10 defines variables starting with the letters A-Z as integers. Line 20 establishes the arrays that will be used in the program. MARKS is the array of skunk mark locations and GRIDARG is the array of grid parameters. Line 30 opens the disk file SCANREC for output, used later.

Line 40 clears the screen and prints a program heading.

Line 60 sets the control option (CTRLOPT) to 4, which means that the scanner will be utilized. Line 70 initializes CMDERR\$, which is the variable describing the command call error status. Line 80 makes the call to the CONTROL routine. Line 90 checks for errors. If the leftmost character in CMDERR\$ is not "@", then an error has occurred and the error number is printed and the program stops. If the leftmost character in CMDERR\$ is "@", the program continues.

```
10 DEFINT A-Z
20 DIM MARKS (2),GRIDARG (7)
30 OPEN "SCANREC" FOR OUTPUT AS #1

40 CLS: PRINT TAB (20) "SAMPLE
PROGRAM FOR NCS/IBM SCANNER
COMMANDS"

60 CTRLOPT=4
70 CMDERR$=SPACE$(20)
80 CALL CONTROL (CTRLOPT,CMDERR$)
90 IF LEFT$(CMDERR$,1)<>"@" THEN
PRINT "CONTROL ERROR -";
CMDERR$:STOP
```

```
110 BAUD =9600
120 PARITY = ASC("0")
130 DATABITS = 7
140 STOPBITS = 2
150 PORTSEL = 1
160 CMDERR$ = SPACE$(20)
170 CALL SETUP (BAUD,PARITY,DATABITS,
  STOPBITS,PORTSEL,CMDERR$)
180 IF LEFT$(CMDERR$,1)<>"@" THEN
  PRINT "SETUP ERROR -";
  CMDERR$:STOP
```

```
200 OFFSET = -1
210 CMDERR$ = SPACE$(20)
220 CALL LEVEL (OFFSET,CMDERR$)
230 IF LEFT$(CMDERR$,1)<>"@" THEN
  PRINT "LEVEL ERROR -";
  CMDERR$:STOP
```

```
250 DOCNUM = 1
260 CELLS = 18
270 TRACKS = 39
280 NUMMARKS = 2
290 MARKS [0] = 1
300 MARKS [1] = 7
310 CMDERR$ = SPACE$(20)
320 ARGPTR = VARPTR(MARKS(0))
330 CALL SKUNK (DOCNUM, CELLS,
  TRACKS, NUMMARKS, ARGPTR,
  CMDERR$)
340 IF LEFT$(CMDERR$,1)<>"@" THEN
  PRINT "SKUNK ERROR -";
  CMDERR$:STOP
```

```
360 LOCATE 20,20:PRINT "Feed sheet
  into scanner or press ESC";
```

Lines 110 through 180 configure the scanner. Lines 110-160 initialize the SETUP variables. Line 170 makes the call to the SETUP routine. Line 180 checks for SETUP errors.

Lines 200 through 230 lower the scanner read level to accept light marks on forms. Line 200 sets the read level offset to -1 which will lower the read level threshold from 4 to 3. Line 210 initializes CMDERR\$. Line 220 makes the call to the SETUP routine. Line 230 checks for errors.

Lines 250 through 340 set up and make the call to the SKUNK routine. Lines 250 through 320 initialize the variables which describe the form to be defined. Line 330 makes the call to the SKUNK routine. Line 340 checks for call errors.

Line 360 prints the feed sheet message onto the screen starting at line 20, column 20.

Line 370 initializes DOC. Line 380 sets READTYPE to 2 which means a form will be scanned and its record passed to the micro. Line 390 initializes the error status variable, CMDERR\$. Line 400 makes the call to the SCAN routine. If the operator pressed the escape key, error number 504 will be returned in CMDERR\$, line 410 will print "PROGRAM ENDED" and the program will end. Line 420 checks for other call parameter errors.

Line 440 allows data statements to be read starting from line 610. A\$, the variable containing the returned GRIDSTRING, is initialized in line 450.

Lines 460 through 570 comprise a loop that reads the grids on the Computest form and saves the returned response strings in A\$. Line 460 starts the loop. The loop will run 8 times, once for each grid on the form. Lines 470 through 490 comprise a loop that reads in the grid characteristics for each of the 8 grids. After the grid characteristics are read in, lines 500 and 510 initializes GRIDSTR\$ (which returns responses) and CMDERR\$ (the error status variable). Line 520 initializes EDSTAT, the variable which identifies edit errors due to incorrect coding on forms. Lines 530 sets ARGPTR (the pointer to the gridarg array) to the start of the array. Line 540 makes the call to the GRID routine. Line 550 checks for parameter errors. Line 560 adds the resolved grid data to the string variable A\$. Line 570 sends the computer back to line 460 for the next grid.

```
370 DOC = 0
380 READTYPE = 2
390 CMDERR$ = SPACE$(20)
400 CALL SCAN (DOC,READTYPE,CMDERR$)
410 IF LEFT$(CMDERR$,3) ="504" THEN
    PRINT "PROGRAM ENDED":CLOSE:END
420 IF LEFT$(CMDERR$,1)<>"@" THEN
    PRINT "SCAN ERROR -";
    CMDERR$:STOP

440 RESTORE 610
450 A$=""

460 FOR I=1 TO 8
470     FOR J=0 TO 7
480         READ GRIDARG(J)
490     NEXT J
500     GRIDSTR$ = SPACE$(20)
510     CMDERR$ = SPACE$(20)
520     EDSTAT = 0
530     ARGPTR = VARPTR(GRIDARG(0))
540     CALL GRID(ARGPTR, GRIDSTR$,
        EDSTAT, CMDERR$)
550     IF LEFT$(CMDERR$,1)<>"@" THEN
        PRINT "GRID ERROR -";
        CMDERR$:STOP
560     A$=A$+GRIDSTR$
570 NEXT I
```

Line 580 prints the resolved data to file #1. Line 590 prints the resolved grid data string on the screen at location 5,1. Line 590 directs the computer back to line 350 which restarts the entire operation beginning with scanning a sheet. Lines 610 through 680 are data statements corresponding to the read statement in line 480.

```
580 PRINT #1,A$
590 LOCATE 5,1:PRINT A$
600 GOTO 350
610 DATA 2,0,10,1,1,3,12,1
620 DATA 3,1,18,1,1,15,19,1
630 DATA 3,1,18,1,1,25,29,1
640 DATA 3,1,18,1,1,35,39,1
650 DATA 3,1,18,1,1,34,30,1
660 DATA 3,1,18,1,1,24,20,1
670 DATA 3,1,18,14,1,14,10,1
680 DATA 3,1,18,14,1,7,3,1
```


]LIST

```

10  DEFINT A-Z
15  GOSUB 1000
20  DIM MARKS (2),GRIDARG (7)
30  OPEN "SCANREC" FOR OUTPUT AS #1
40  CLS: PRINT TAB (20) "SAMPLE
    PROGRAM FOR NCS/IBM SCANNER
    COMMANDS"
50  REM*****
60  CTRLPT=4
70  CMDERR$=SPACE$(20)
80  CALL CONTROL (CTRLPT,CMDERR$)
90  IF LEFT$(CMDERR$,1)<>"@" THEN
    PRINT "CONTROL ERROR - ";
    CMDERR$:STOP
100 REM*****
110 BAUD =9600
120 PARITY = ASC("O")
130 DATABITS = 7
140 STOPBITS = 2
150 PORTSEL = 1
160 CMDERR$ = SPACE$(20)
170 CALL SETUP (BAUD,PARITY,DATABITS,
    STOPBITS,PORTSEL,CMDERR$)
180 IF LEFT$(CMDERR$,1)<>"@" THEN
    PRINT "SETUP ERROR - ";
    CMDERR$:STOP
190 REM*****
200 OFFSET = -1
210 CMDERR$ = SPACE$(20)
220 CALL LEVEL (OFFSET,CMDERR$)
230 IF LEFT$(CMDERR$,1)<>"@" THEN
    PRINT "LEVEL ERROR -";
    CMDERR$:STOP
240 REM*****
250 DOCNUM = 1
260 CELLS = 18
270 TRACKS = 39
280 NUMMARKS = 2
290 MARKS (0) = 1
300 MARKS (1) = 7
310 CMDERR$ = SPACE$(20)
320 ARGPTR = VARPTR(MARKS(0))
330 CALL SKUNK (DOCNUM,CELLS,TRACKS,
    NUMMARKS,ARGPTR,CMDERR$)

```

```

340 IF LEFT$(CMDERR$,1)<>"@" THEN
    PRINT "SKUNK ERROR -";
    CMDERR$:STOP
350 REM*****
360 LOCATE 20,20:PRINT "Feed sheet
    into scanner or press ESC";
370 DOC = 0
380 READTYPE = 2
390 CMDERR$ = SPACE$(20)
400 CALL SCAN (DOC,READTYPE,CMDERR$)
410 IF LEFT$(CMDERR$,3) ="504" THEN
    PRINT PROGRAM ENDED":CLOSE:END
420 IF LEFT$(CMDERR$,1)<>"@" THEN
    PRINT "SCAN ERROR -";
    CMDERR$:STOP
430 REM*****
440 RESTORE 610
450 A$=""
460 FOR I=1 TO 8
470     FOR J=0 TO 7
480         READ GRIDARG(J)
490     NEXT J
500     GRIDSTR$ = SPACE$(20)
510     CMDERR$ = SPACE$(20)
520     EDSTAT = 0
525     Y#=FRE(0)
530     ARGPTR = VARPTR(GRIDARG(0))
540     CALL GRID(ARGPTR, GRIDSTR$,
        EDSTAT,CMDERR$)
550     IF LEFT$(CMDERR$,1)<>"@" THEN
        PRINT "GRID ERROR -";
        CMDERR$:STOP
560     A$=A$+GRIDSTR$
570 NEXT I
580 PRINT #1,A$
590 LOCATE 5,1:PRINT A$
600 GOTO 350
610 DATA 2,0,10,1,1,3,12,1
620 DATA 3,1,18,1,1,15,19,1
630 DATA 3,1,18,1,1,25,29,1
640 DATA 3,1,18,1,1,35,39,1
650 DATA 3,1,18,1,1,34,30,1
670 DATA 3,1,18,14,1,14,10,1
680 DATA 3,1,18,14,1,7,3,1

```

INTERPRETIVE BASIC

```
1000 DEFSEG = $H1D54
1010 BLOAD "COMNDS.BAS",0
1020 SCAN = PEEK(0)+256*PEEK(1)
1030 GRID = PEEK(2)+256*PEEK(3)
1040 SKUNK = PEEK(4)+256*PEEK(5)
1050 CONTROL = PEEK(6)+256*PEEK(7)
1060 SETUP = PEEK(8)+256*PEEK(9)
1070 LEVEL = PEEK(10)+256*PEEK(11)
1080 TRANSMIT = PEEK(12)+256*PEEK(13)
1090 RECV = PEEK(14)+256*PEEK(15)
1100 RETURN
```

6

PROGRAM
ERROR
CODES

Introduction.....6-2
Errors.....6-3

INTRODUCTION

The NCS Micro Scanner Commands are designed to be easily incorporated into application programming. To facilitate easy use of the commands, NCS has designed a set of meaningful error codes for each command. Error codes are divided into eight different levels:

- 100 level - CONTROL Errors
- 200 level - SETUP Errors
- 300 level - GRID Errors
- 400 level - SKUNK Errors
- 500 level - SCAN Errors
- 600 level - LEVEL Errors
- 700 level - TRANSMIT Errors
- 800 level - RECV Errors

OVERVIEW: ERRORS

If an error occurs during the operation of Micro Scanner Commands, it will be noted and passed back to the application program in the cmderr variable.

CONSIDERATIONS: ERRORS

Since programming error codes are passed back to the application program within the variable cmderr, the programmer can determine the effect of errors on the program. The programmer can choose to stop the program in the case of a serious error. Or, if an inconsequential error occurs, the programmer can choose to print out an error flag while continuing on in the program.

For a description of variable names which are referred to in this error section, see the appropriate command in Section Three.

PROGRAMMING ERRORS: CONTROL, SETUP

NUMBER	PROBLEM	EXPLANATION
101	Incorrect scanner control option number (<1 or >4)	Ctrlopt has been assigned an incorrect value. Ctrlopt cannot be less than 1 or greater than 4.
102	Scanner device not ready to receive transmission.	The operator has pressed the ESC key and the scanner device is not ready to receive the transmission. Check connections and modem status. Then retransmit the data.
201	Incorrect speed indication (baud rate)	The baud variable (which describes the baud rate) has been assigned an incorrect value. Correct values are 110, 300, 600, 1200, 2400, 4800, and 9600.
202	Incorrect parity indication	The parity variable has been assigned an incorrect value. Correct values are the ASCII values for O, E, and N.
203	Incorrect number of data bits indicated	The databits variable (which describes the number of bits per character) has been assigned an incorrect value. Correct values are 7 and 8
204	Incorrect number of stop bits indicated	The stopbits variable has been assigned an incorrect value. Correct values are 1 and 2.
205	Incorrect board selection indicated	The variable portsel has been assigned an incorrect value. Correct values for portsel are 1 and 2.
206	Incorrect combination of parameters	This error occurs if the user tries to set 8 data bits, parity and 2 stop bits, which is a total of 11 bits per character. The maximum allowed is 10 bits per character.

PROGRAMMING ERRORS: GRID

NUMBER	PROBLEM	EXPLANATION
301	Incorrect type indication (Gridarg(0))	The grid type has been incorrectly defined or an attempt was made to link two different types of grids. The variable type cannot be less than 1 or greater than 7. If this grid has been linked to another grid, ensure that they are the same types.
302	Incorrect starting x position (Gridarg(2))	The start x variable, sx, has been incorrectly defined. One of two conditions exists: 1. The start x position is out of the range indicated in the SKUNK command. For example: cells = 35 start x = 40 If there are 35 cells, start x must be from 1 to 35. 2. The start x position was not defined or start x was defined but was not inserted into the GRID command call.
303	Incorrect ending x position (Gridarg(3))	The end x variable, ex, has been incorrectly defined. One of two conditions exists: 1. The end x position is out of the range indicated in the SKUNK command. For example: cells = 37 end x = 41 If there are 37 cells, end x must be from 1 to 37. 2. The end x position was not defined or end x was defined but was not inserted into the GRID command.

PROGRAMMING ERRORS: GRID		
NUMBER	PROBLEM	EXPLANATION
304	X spacing out of range indicated in SKUNK command (Gridarg(4))	<p>X Spacing is out of range given the definition of the variable cells in the SKUNK table. For example:</p> <p style="text-align: center;">cells = 43 spacing = 47</p> <p>Since there are only 43 cells, spacing cannot be 47.</p>
305	Incorrect starting y position (Gridarg(5))	<p>The start y variable, sy, has been incorrectly defined. One of two conditions exists:</p> <ol style="list-style-type: none"> 1. The start y position is out of the range indicated in the SKUNK command. For example: <p style="text-align: center;">tracks = 32 start y = 34</p> <p>If there are 32 timing marks, start y must be from 1 to 32.</p> 2. The start y position was not defined or sy was defined but was never inserted into the GRID command call.
306	Incorrect ending y position (Gridarg(6))	<p>The end y variable, ey, has been incorrectly defined. One of two conditions exists:</p> <ol style="list-style-type: none"> 1. The end y position is out of the range indicated in the SKUNK command. For example: <p style="text-align: center;">tracks = 30 end y = 37</p> <p>If there are 30 timing marks, end y must be from 1 to 30.</p> 2. The end y position was not defined or end y was defined but was not inserted into the GRID command.

PROGRAMMING ERRORS: GRID

NUMBER	PROBLEM	EXPLANATION
307	Y spacing out of range indicated in SKUNK command (Gridarg(7))	<p>Y spacing is out of range given the definition of the variable tracks in the SKUNK table. For example:</p> <p style="text-align: center;">tracks = 30 spacing = 31</p> <p>Since tracks is only 30, spacing cannot be 31.</p>
308	Spacing x is impossible.	<p>Given the starting and ending x coordinates, the spacing between x response positions is incorrect. For example:</p> <p style="text-align: center;">start x = 8 end x = 15 spacing x = 2</p> <p>If the first x response position is 8, the next will be 10, the next 12, the next 14, etc. The last x response position could <u>not</u> be an odd number.</p>
309	Spacing y is impossible.	<p>Given the starting and ending y coordinates, the spacing between y response positions is incorrect. For example:</p> <p style="text-align: center;">start y = 7 end y = 15 spacing y = 3</p> <p>If the first y response position is 7, the next will be 10, the next 13, the next 16, etc. End y could <u>not</u> be 15.</p>
314	Class is out of range (Gridarg(1))	<p>Class cannot be less than 0 or greater than 7. Class describes whether the grid is vertical, horizontal, check parity, or linked grid.</p>

PROGRAMMING ERRORS: GRID, SKUNK

NUMBER	PROBLEM	EXPLANATION
321	Number responses per item wrong for grid type or number of items exceeds maximum allowed.	Given the grid type, the number of responses per item is wrong. For example, a grid is deemed a numeric grid. However, coordinates indicate that there are 36 responses per item. A numeric grid must contain no more than 10 responses per item (0-9). The only grid types with 36 possible responses per item are alphanumeric grids and two-digit response grids. In the case of binary grids, a maximum of 28 items is allowed per grid. A linked grid can only resolve one item.
322	Number of items exceeded in linked grid.	More than one item was resolved in a grid with a link class.
400	Document number out of range (1-99)	The variable docnum is out of range. Docnum cannot be less than 1 or greater than 99.
401	Value for cells out of range (1-47)	The variable cells, the highest X response position, is out of range. Cells cannot be less than 1 or greater than 47.
402	Value for number of timing tracks out of range (1-99)	The variable tracks, the number of timing tracks on a form, is out of range. Tracks cannot be less than 1 or greater than 99.
403	Number of SKUNK mark positions out of range (1-47)	The variable nummarks, the number of SKUNK mark positions occupied, is out of range. Nummarks cannot be less than 1 or greater than 47.
404	A value for a skunk mark position is out of range (1-47)	A value found in the array marks, a list of the response positions occupied by skunk marks, is out of range. The value for marks cannot be less than 1 or greater than 47.

PROGRAMMING ERRORS: SKUNK, SCAN

NUMBER	PROBLEM	EXPLANATION
405	A value for a skunk mark position is repeated	A value found in the array marks is repeated. For example, in the array marks (1,2,7,7,15) the position 7 is repeated. Since there is only one position 7, it should be listed just once.
406	Two documents have the same skunk marks	Two documents have the same skunk marks. In other words, the programmer has defined a document which is already defined with another docnum. The first document definition will not be altered.
407	Number of Skunk entries greater than maximum allowed	More than the allotted 99 entries have been entered into the skunk table.
504	Operator pressed the "ESC" key	The operator pressed the "ESC" key while the SCAN command was operating.
505	Invalid SCAN command calling option	The SCAN command calling option found in readtype is incorrect. The calling options are: <ul style="list-style-type: none"> ●2 - request new document from scanner ●3 - request retransmission of current record from scanner
507	Skunk marks do not match any document in skunk table.	A document image has been received and is in the buffer but the skunk marks do not match any document which has been defined in the skunk table.

PROGRAMMING ERRORS: SCAN

NUMBER	PROBLEM	EXPLANATION
508	Document match made but more or less data than defined in SKUNK	Although the document is matched up with a document definition in the skunk table (via the defined skunk marks), there is more or less data than defined in the skunk command. For example, a document could be matched through skunk marks, but the sheet could become jammed and less than a complete record would be transferred to the host. In this case the number of timing marks for the jammed sheet would be more or less than the number of timing marks defined in the SKUNK command.
509	Sheet buffer overflow	Data for more than 99 timing tracks x 48 cells were received. This could be a hardware problem or the communications protocol is incorrect.
510	Illegal compression count received	When a record contains four or more identified characters in a row the data is compressed as it is transmitted from the scanner to the microcomputer. The compression is not working correctly. This error occurs when the communications protocol is incorrect. The compression character code on the configuration sheet should be checked.
511	Sheet buffer overflow while decompressing sheet buffer	As data is being decompressed, the sheet buffer overflows. This could be a hardware problem or communications protocol could be incorrect.
512:x	<p>Communications link error</p> <p>x = 01 for overrun error</p> <p>x = 02 for parity error</p> <p>x = 04 for framing error</p>	<p>One of the following communications errors occurs: data is received too fast, parity is in error, timing of reception is off. These are all hardware problems. If this error is a recurring problem, the communications protocol of the scanner does not match that of the host. If the error occurs occasionally it is an electrical noise problem. Check the cable to see that it does not lie next to heavy extension cords or near other office equipment.</p>

PROGRAMMING ERRORS: LEVEL, TRANSMIT

NUMBER	PROBLEM	EXPLANATION
601	Incorrect offset value	The value of offset, the variable describing the offset value, is incorrect. Offset should be from -2 to 2.
701	Incorrect device number	<p>The number selected for dest (DEST*) is incorrect. Dest indicates the device to which outputs is to be directed:</p> <ul style="list-style-type: none"> ●0 = scanner transport printer ●1 = scanner auxiliary port ●2 = scanner LED display <p>The numbers 0, 1 and 2 are the only possible values for dest (DEST*).</p>
702	Incorrect transport printer start position (< 1 or > 99)	The start print position for the transport printer is incorrect. Valid start print positions are from 1 to 99.
703	Number of output characters > device limit or value of output characters does not fall within range limit	<p>One of two conditions exists:</p> <ul style="list-style-type: none"> ●The number of output characters is beyond the device limits. For example, if the programmer wishes to output a string of 22 characters in length the programmer would not choose the LED display. Since the LED display is limited to one character, 22 characters would be beyond the LED display limits. ●The value of the output character does not fall within the range limit. For example, the LED display can display only 1-9 and A-F. If the letter M were received by the LED display, the error code 703 would be returned by cmderr.
704	Escape key pressed	Operator pressed the Escape key. The TRANSMIT command is terminated whether or not any transmission has taken place. This can be useful if the scanner and host do not seem to be communicating.

PROGRAMMING ERRORS: RECV

NUMBER	PROBLEM	EXPLANATION
801	ESC key pressed during RECV call	The ESC key was pressed while the RECV routine was in operation.
802:x	Communications error detected x = 1 for overrun error x = 2 for parity error x = 3 for framing error	The data was not recognized as data or was received incorrectly. Check to ensure that the communications cable is hooked up correctly.



APPENDIX B

TEST
PROGRAM

TEST PROGRAM.....B-1
Introduction.....B-2
Test Program.....B-3
Compiled Basic Test
 Program.....B-13
Interpretive Basic Test
 Program.....B-19
Pascal Test Program.....B-23

INTRODUCTION

This appendix describes the scanner commands sample testing program and lists the program in Compiled Basic, Interpretive Basic, and Pascal.

OVERVIEW: TEST PROGRAM

A sample program is on the scanner commands diskette in an executable format for testing scanner commands with actual data to ensure that parameters are being correctly passed to command routines and returned to the host program. The test routine is also useful in determining whether communications problems exist. The test routine is easy to use since data can be entered via the keyboard.

The test programs included on the diskette are the following versions:

Compiled Basic (CTESTER.EXE)
 Interpretive Basic (ITESTER.BAS)
 Pascal (PTESTER.BAS)

EXPLANATION: TEST PROGRAM

A run through of the program follows with illustrations of the micro screens and instructions of how to proceed.

- To utilize the testing program from the Basic Operating System, enter one of the following:

B:CTESTER for Compiled Basic
 B:PTESTER for Pascal
 B:ITESTER for Interpretive Basic

- Then press the enter key.

1. CONTROL COMMAND	4. SKUNK COMMAND	7. TRANSMIT COMMAND
2. SETUP COMMAND	5. SCAN COMMAND	8. RECV COMMAND
3. GRID COMMAND	6. LEVEL COMMAND	9. DISP. SHEET OR SKUNK TABLE
10. QUIT		
ENTER SELECTION (1..10) FROM ABOVE ?		

TESTING PROGRAM MAIN MENU

- Enter the number of the command you want to test (1..8) or enter 9 to display the sheet buffer or skunk table. (Option 9 is only available in the Compiled Basic program.)

- Then press the enter key.

CAUTION: A record must be passed (through the SCAN command) before the CONTROL, TRANSMIT, and RECV commands will be effective.

TEST PROGRAM

EXPLANATION: TEST PROGRAM (cont.)

Control Command

This message appears when CONTROL COMMAND is selected from the main menu:

```
TESTING CONTROL COMMAND

1. RELEASE DOCUMENT      3. SELECT AUX PORT
2. STOP SCANNER         4. SELECT SCANNER

ENTER SELECTION FROM ABOVE (1..4)
?
```

- Press the number of the desired option (1..4). Then press the enter key.

```
ERROR STRING:           ERROR STRING :
@                       101
```

If the transmission has been made correctly, the message on the left will be returned. The cmderr variable, which returns a parameter error string, contains "@", indicating no error has been made.

If an error or errors have occurred, the message on the right will appear listing parameter error numbers. These errors are explained in Section Six.

```
DO AGAIN (Y/N) : ?
```

After each command routine is tested, this message appears.

- If the test is to be repeated, enter Y and press the enter key.
- If the test is not to be repeated, enter N and press the enter key. The system will return to the main menu.

EXPLANATION: TEST PROGRAM (cont.)Setup Command

This message appears when SETUP COMMAND is selected from the main menu:

```

                                TESTING SETUP COMMAND

                                ENTER BAUD RATE :?
                                ENTER PARITY    :?
                                ENTER DATA BITS :?
                                ENTER STOP BITS  :?
                                ENTER BOARD SELECT :?

```

- Enter the baud rate, parity, data bits, stop bits, and board selection. Press the enter key after each entry.

Just as described in the CONTROL COMMAND selection, an "@" will be returned if there are no parameter errors. If there are parameter errors, they will be listed.

GRID Command

This message appears when GRID COMMAND is selected from the main menu:

```

                                TESTING GRID COMMAND

                                INPUT TYPE  ?
                                INPUT CLASS ?
                                INPUT SX    ?
                                INPUT EX    ?
                                INPUT IX    ?
                                INPUT SY    ?
                                INPUT EY    ?
                                INPUT IY    ?

```

- Enter the type, class, start x position, end x position, x spacing, start y position, end y position, and spacing y. Press the enter key after each entry.

TEST PROGRAM

EXPLANATION: TEST PROGRAM (cont.)

GRID Command (cont.)

```
GRID STRING:  
EDIT STATUS:  
ERROR STRING:  
@004
```

The system returns the grid string, then @ followed by the number of characters in the gridstring if no errors have occurred. If parameter errors occur, EDIT STATUS will be 128 and ERROR STRING will list the parameter error numbers.

Note: Errors will occur if the GRID command is tested before entering the document parameters under the SKUNK command. Make certain to test the GRID command after defining forms via the SKUNK command.

SKUNK Command

This message appears when SKUNK COMMAND is selected from the main menu:

```
TESTING SKUNK COMMAND  
  
INPUT DOC ?  
INPUT CELLS ?  
INPUT TRACKS?  
INPUT MARKS ?  
INPUT MARK LOCATION, 99 TO EXIT :?  
INPUT MARK LOCATION, 99 TO EXIT :?
```

- Enter the document number, number of cells, number of timing tracks, number of skunk marks, and locations of skunk marks. Press the enter key after each entry.
- Enter the number 99 to indicate that all skunk mark locations are listed. The system will keep asking for skunk mark locations until 99 is entered.

As with the other commands, parameter errors are listed if they occur.

EXPLANATION: TEST PROGRAM (cont.)SCAN Command

This message appears when SCAN COMMAND is selected from the main menu:

```
TESTING SCAN COMMAND
ENTER 2 FOR SCAN, 3 FOR RE-TRANSMIT :?
```

- Enter the number 2 for scan or 3 for re-transmit. (Remember that a document must first be scanned before it can be re-transmitted.) Then press the enter key.

```
I.D. NUMBER OF DOCUMENT SCANNED :
ERROR STRING:
```

The document I.D. number will be returned along with parameter errors, if they occur.

Note: Again, this command should not be used unless the form to be scanned has been defined with the SKUNK command.

LEVEL Command

This message appears when LEVEL COMMAND has been selected from the main menu:

```
TESTING LEVEL COMMAND
ENTER OFFSET TO READ LEVEL (-2..+2)
```

- Enter the number of the read level adjustment and press the enter key.

```
ERROR STRING:
@5
```

Again, parameter errors are listed if they occur. If no parameter errors occur, the adjusted read level will be returned in cmderr along with the @ symbol.

TEST PROGRAM

EXPLANATION: TEST PROGRAM (cont.)

TRANSMIT Command

This message appears when TRANSMIT COMMAND is selected from the main menu:

TESTING TRANSMIT COMMAND
ENTER DESTINATION (0=PRINTER,1=AUXPORT,2=LED):?

- Enter the number indicating the data destination and press the enter key.

ENTER VALUE (0..99) START PRINT POSITION :?
ENTER STRING TO TRANSMIT:?

- If 0 (printer) is selected, enter the start print position. Then press the enter key.
- Then enter the string to be transmitted. Press the enter key to signal the end of the string. (Do not enter , or : in the Basic test program as either mark is interpreted as a signal to end the string.)

ENTER STRING TO TRANSMIT:?

- If 1 (auxport) is selected, enter the string to transmit. Then press the enter key.

ENTER VALUE (0..15) TO BE DISPLAYED:?

- If 2 (LED-scanner display panel) is selected, enter the value to be displayed. Then press the enter key.

In all three cases, parameter errors will be listed if they occur.

TEST PROGRAM

EXPLANATION: TEST PROGRAM (cont.)

RCV Command

This message appears when RCV COMMAND is selected from the main menu:

TESTING RCV COMMAND

ENTER STRING AT AUX PORT TERMINAL AND PRESS
RETURN AT TERMINAL TO TERMINATE OR ESCAPE ON PC TO TERMINATE
INPUT STRING:

- Enter the string at the aux port terminal. Press the enter key to terminate the string.

If parameter errors occur, they will be listed.

DISPLAY SHEET OR SKUNK TABLE

The sheet buffer or skunk table can be displayed in the Compiled Basic Testing Program. This message appears when DISPLAY SHEET OR SKUNK TABLE is selected from the main menu:

DISPLAY SHEET BUFFER OR SKUNK TABLE

1. DISPLAY SHEET BUFFER
2. DISPLAY SKUNK TABLE
3. RETURN TO MAIN MENU

ENTER SELECTION FROM ABOVE (1..3): ?

DISPLAY MENU

- Enter the number of the desired option and press the enter key.

If option 1, DISPLAY SHEET BUFFER, is selected, this message appears:

DISPLAY SHEET BUFFER

ENTER DESIRED ROW (1..99) OF SHEET TO BE DISPLAYED: ?

- Enter the number of the row to start displaying at. Then press the enter key. Eleven rows will be displayed at one time.

TEST PROGRAM

EXPLANATION: TEST PROGRAM (cont.)

ROW	LOCATION	READ LEVELS
1	1	7170003006611272225575100000030000711131777004
2	49	00000600603222210000706066100040007671000506115
3	97	22222000600006700061114000000050001110660070605
4	145	00000000000010000010000000000406660070200307774
5	193	22222322222000500050077711100600101700300000404
6	241	00000000007000000061113166000700003020020500564
7	289	0001000607007011140100000000000000050000500075
8	337	77000000500060606001111000000000050000660000604
9	385	00066000021000000650000113100000700006000030005
10	433	0707111111000000000111100000607000001216660005
11	481	000000000000010000060060611160430000000006004

DO AGAIN ? (Y/N): ?

- To display another set of rows, press Y and follow the same procedure.
- To get back to the display menu, press N and press the enter key.

If option 2, DISPLAY SKUNK TABLE, is selected, this message appears:

DISPLAY SKUNK TABLE

ENTER DESIRED LOCATION IN TABLE (1..99): ?

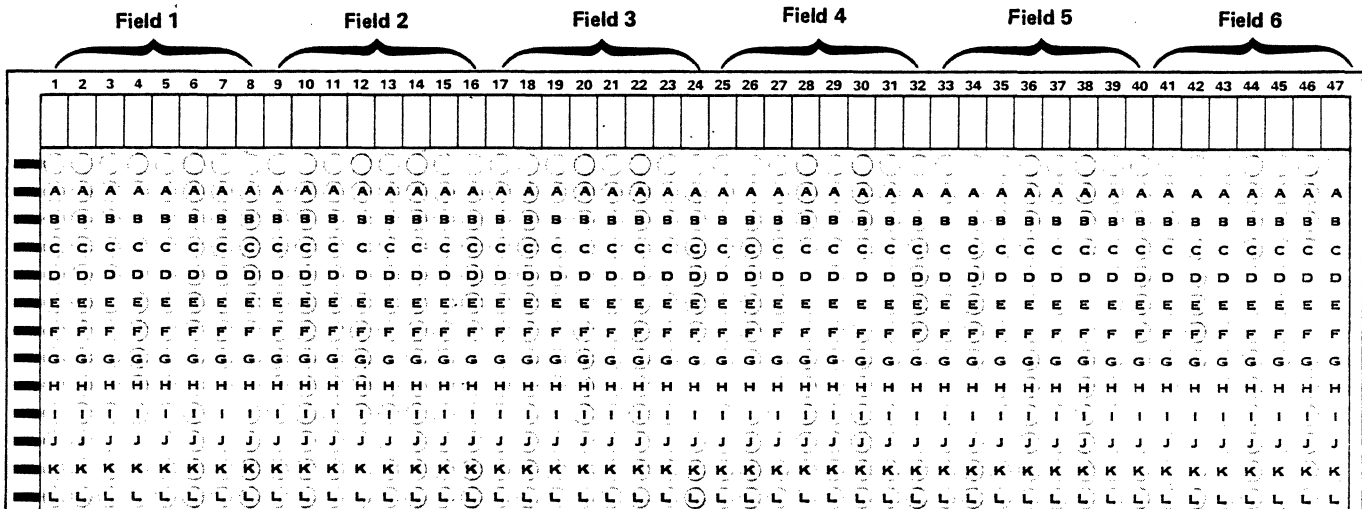
- Enter the number of the entry to start displaying at. Then press the enter key. Eleven entries will be displayed at one time.

EXPLANATION: TEST PROGRAM (cont.)

ENTRY	DOC	CELLS	TRACKS	NMARKS	MARKS	1..47
1	1	48	65	2	2	8 0 0 0
2	2	48	65	3	8	24 0 0 0
3	3	48	65	3	0	0 26 0 0
4	4	48	65	2	16	44 0 0 0
5	5	48	65	4	48	24 32 0 0
6	6	48	65	3	2	96 0 0 0
7	7	48	65	3	8	0 32 0 0
8	8	48	65	2	0	0 0 192 0
9	9	48	65	1	16	0 0 0 0
10	10	48	65	4	20	48 0 0 0
11	11	48	65	2	0	0 96 0 0

DO AGAIN ? (Y/N) :
?

When programming with the SKUNK command, skunk mark locations are entered by matching their X locations on the form. In the skunk buffer, the marks are stored in a different manner. The following illustration and description explains how skunk mark locations are represented in the skunk buffer:



Notice that there are 47 possible skunk mark locations on the form. The form is physically divided into 6 fields. Each field contains 8 response positions except field 6, which contains 7 response positions. Field 1 contains X locations 1-8; field 2 contains X location 9-16, etc.

TEST PROGRAM

EXPLANATION: TEST PROGRAM (cont.)

Each field (1..6) contains 8 possible values. Position values are expressed in powers of two. So positions 1 through 8 represent the values 1(2^0), 2(2^1), 4(2^2), 8(2^3), 16(2^4), 32(2^5), 64(2^6), and 128(2^7).

	Field 1								Field 2							
Positions	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Values	1	2	4	8	16	32	64	128	1	2	4	8	16	32	64	128

The powers of 2 numbering sequence is repeated for each field so that the same 8 values are repeated for the 8 positions of field 2, field 3, field 4, field 5, and field 6.

When reviewing skunk mark locations in the skunk buffer, a value appears for each of the 6 fields. For example:

skunk marks
in these
positions

	8								24								0								0								0								0							
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47		
			X								X	X																																				
A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	
B	B	B	B	B	B	B	B	B	B	B	B	B	B	B	B	B	B	B	B	B	B	B	B	B	B	B	B	B	B	B	B	B	B	B	B	B	B	B	B	B	B	B	B	B	B	B	B	
C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	
D	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D	
E	E	E	E	E	E	E	E	E	E	E	E	E	E	E	E	E	E	E	E	E	E	E	E	E	E	E	E	E	E	E	E	E	E	E	E	E	E	E	E	E	E	E	E	E	E	E	E	
F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	
G	G	G	G	G	G	G	G	G	G	G	G	G	G	G	G	G	G	G	G	G	G	G	G	G	G	G	G	G	G	G	G	G	G	G	G	G	G	G	G	G	G	G	G	G	G	G	G	
H	H	H	H	H	H	H	H	H	H	H	H	H	H	H	H	H	H	H	H	H	H	H	H	H	H	H	H	H	H	H	H	H	H	H	H	H	H	H	H	H	H	H	H	H	H	H	H	
I	I	I	I	I	I	I	I	I	I	I	I	I	I	I	I	I	I	I	I	I	I	I	I	I	I	I	I	I	I	I	I	I	I	I	I	I	I	I	I	I	I	I	I	I	I	I	I	

In this example, three skunk marks are found on a form in locations 4, 12, and 13. Since location 4 has a value of 8, and 8 appears in the skunk table for the value of field 1. Positions 12 and 13 also contain skunk marks. Their values are 8 and 16, which add up to 24. 24 appears in the skunk table for the value of field 2. Since fields 3 through 6 contain no skunk marks, zeros appear in those positions in the skunk table.

- To display another set of entries, press Y and follow the same procedure.
- To get back to the display menu, press N.

Press 3 and the enter key if you want to return to the main menu.

QUIT

When QUIT is selected from the main menu, you will return to the Basic Operating System

EXPLANATION: TEST PROGRAM

The actual test program is listed in its entirety in Compiled Basic, Interpretive Basic, and Pascal as a further illustration of how to incorporate scanner commands into host programming. The first listing is in Compiled Basic.

```

50 REM  FILENAME: CTESTER   COMPILED BASIC TEST ROUTINE
55 REM  FILE CREATED: 28-JUN-83
60 REM  LAST REVISED: 28-JUN-83  10:00
65 REM  CONTEXT: TESTE PROGRAM FOR COMPILED BASIC VERSION OF MICROSCANNER CMDS
70 REM
71 REM  THIS TEST PROGRAM ALLOWS THE USER TO DISPLAY THE SHEET BUFFER AND
72 REM  SKUNK TABLE.  TEST PROGRAMS FOR OTHER LANGUAGES (INTERPRETIVE BASIC
73 REM  AND IBM PASCAL) DO NOT HAVE THIS FEATURE.
74 REM
75 REM  NOTE: THE LINES THAT ARE COMMENTED OUT ARE USED IN THE INTERPRETIVE
76 REM        BASIC TEST PROGRAM ( LINES 100-190)
77 REM
78 REM
100 'DEF SEG=&H1CF4
110 'BLOAD "cmds.bas",0
120 'SCAN =PEEK(0) + 256 + PEEK(1)
130 'GRID =PEEK(2) + 256 + PEEK(3)
140 'SKUNK =PEEK(4) + 256 + PEEK(5)
150 'CONTROL =PEEK(6) + 256 + PEEK(7)
160 'SETUP  =PEEK(8) + 256 + PEEK(9)
170 'LEVEL  =PEEK(10) + 256 + PEEK(11)
180 'TRANSMIT=PEEK(12) + 256 + PEEK(13)
190 'RECV   =PEEK(14) + 256 + PEEK(15)
200 DIM GRIDARGZ(8)
210 DIM MARKSZ(48)
220 REM*****
230 REM    MENU FOR TEST PROGRAM
240 REM*****
1000 PRINT "1. CONTROL COMMAND      4. SKUNK COMMAND      7. TRANSMIT COMMAND"
1010 PRINT "2. SETUP COMMAND        5. SCAN COMMAND       8. RECV COMMAND "
1020 PRINT "3. GRID COMMAND              6. LEVEL COMMAND     9. DISP. SHEET OR SKUNK TABLE"
1030 PRINT: PRINT "10. QUIT"
1040 PRINT: PRINT "ENTER SELECTION (1..10) FROM ABOVE: ";
1050 INPUT AZ
1055 IF AZ >10 THEN GOTO 1000
1060 IF AZ=10 THEN SYSTEM

```

COMPILED BASIC TEST PROGRAM

```
1130 ON AZ GOSUB 10000,11000,12000,13000,14000,15000,16000,17000,18000
1140 GOTO 1000
10000 REM*****
10005 REM SUBROUTINE TO TEST CONTROL COMMAND
10007 REM*****
10010 CLS
10020 PRINT "TESTING CONTROL COMMAND":PRINT
10030 PRINT "1. RELEASE DOCUMENT      3. SELECT AUX PORT"
10040 PRINT "2. STOP SCANNER          4. SELECT SCANNER"
10050 PRINT: PRINT "ENTER SELECTION FROM ABOVE (1..4)"
10060 INPUT CTRLPTZ
10070 CMDERR$=SPACE$(20)
10080 CALL CONTROL(CTRLPTZ,CMDERR$)
10090 PRINT "ERROR CODE: ";CMDERR$
10100 PRINT "DO AGAIN? (Y/N)";:INPUT DOAGAIN$
10110 IF DOAGAIN$ <> "N" THEN GOTO 10020
10120 RETURN
11000 REM*****
11005 REM SUBROUTINE TO TEST SETUP COMMAND
11007 REM*****
11010 CLS
11020 PRINT "TESTING SETUP COMMAND":PRINT
11030 PRINT "ENTER BAUD RATE: ";:INPUT BAUDZ
11040 PRINT "ENTER PARITY(O,E,N)";:INPUT PARITY$
11050 PRINT "ENTER DATA BITS: ";:INPUT DATABITZ
11060 PRINT "ENTER STOP BITS: ";:INPUT STOPBITZ
11070 PRINT "ENTER BOARD SELECT: ";:INPUT PORTZ
11080 PARITYZ=ASC(PARITY$)
11085 CMDERR$=SPACE$(200)
11090 CALL SETUP(BAUDZ,PARITYZ,DATABITZ,STOPBITZ,PORTZ,CMDERR$)
11100 PRINT "ERROR STRING: ",CMDERR$
11110 PRINT "DO AGAIN? (Y/N)";:INPUT DOAGAIN$
11120 IF DOAGAIN$ <> "N" THEN GOTO 11020
11130 RETURN
12000 REM*****
12005 REM SUBROUTINE TO TEST GRID ROUTINE
12007 REM*****
12010 CLS
12015 PRINT "TESTING GRID COMMAND":PRINT
12020 PRINT "INPUT TYPE";:INPUT GRIDARGZ(0)
12030 PRINT "INPUT CLASS";:INPUT GRIDARGZ(1)
12040 PRINT "INPUT SX";:INPUT GRIDARGZ(2)
12050 PRINT "INPUT EX";:INPUT GRIDARGZ(3)
12060 PRINT "INPUT IX";:INPUT GRIDARGZ(4)
12070 PRINT "INPUT SY";:INPUT GRIDARGZ(5)
12080 PRINT "INPUT EY";:INPUT GRIDARGZ(6)
12090 PRINT "INPUT IY";:INPUT GRIDARGZ(7)
12100 Y#=FRE(0)
12110 EDSTATZ=99
```

```

12120 GRIDSTR$=SPACE$(254):CMDERR$=SPACE$(200)
12130 ARGPTRZ= VARPTR(GRIDARGZ(0))
12140 CALL GRID(ARGPTRZ,GRIDSTR$,EDSTATZ,CMDERR$)
12150 PRINT "GRID STRING:",GRIDSTR$:PRINT "EDIT STATUS:",EDSTATZ
12160 PRINT "ERROR STRING:",CMDERR$
12180 PRINT "DO AGAIN? (Y/N):";:INPUT DOAGAIN$
12190 IF DOAGAIN$ <> "N" THEN GOTO 12020
12200 RETURN
13000 REM*****
13005 REM SUBROUTINE TO TEST SKUNK COMMAND
13007 REM*****
13010 CLS
13020 PRINT "TESTING SKUNK COMMAND":PRINT
13030 PRINT "INPUT DOC#:";:INPUT DOCNUMZ
13040 PRINT "INPUT CELLS:";:INPUT CELLSZ
13050 PRINT "INPUT TRACKS:";:INPUT TRACKSZ
13060 PRINT "INPUT NMARKS:";:INPUT NUMMARKSZ
13070 FOR K=0 TO 50 : PRINT "INPUT MARK LOCATION,99 TO EXIT:";
13075 INPUT MARKSZ(K)
13080 IF MARKSZ(K)=99 THEN GOTO 13100
13090 NEXT K
13100 CMDERR$=SPACE$(200)
13110 Y#=FRE(0)
13120 MARKPTRZ=VARPTR(MARKSZ(0))
13130 CALL SKUNK(DOCNUMZ,CELLSZ,TRACKSZ,NUMMARKSZ,MARKPTRZ,CMDERR$)
13140 PRINT "ERROR STRING:",CMDERR$
13150 PRINT "DO AGAIN? (Y/N):";:INPUT DOAGAIN$
13160 IF DOAGAIN$ <> "N" THEN GOTO 13020
13170 RETURN
14000 REM*****
14005 REM SUBROUTINE TO TEST SCAN ROUTINE
14007 REM*****
14010 CLS
14020 PRINT "TESTING SCAN COMMAND":PRINT
14030 PRINT "ENTER 2 FOR SCAN, 3 FOR RE-TRANSMIT:";:INPUT READTYPEZ
14040 CMDERR$=SPACE$(200):DOCZ=0
14050 CALL SCAN(DOCZ,READTYPEZ,CMDERR$)
14060 PRINT "I.D. NUMBER OF DOCUMENT SCANNED: ";
14065 PRINT USING "##"; DOCZ
14070 PRINT "ERROR STRING:",CMDERR$
14080 PRINT "DO AGAIN? (Y/N):";:INPUT DOAGAIN$
14090 IF DOAGAIN$ <> "N" THEN GOTO 14020
14100 RETURN
15000 REM*****
15005 REM SUBROUTINE TO TEST LEVEL COMMAND
15007 REM*****
15008 CLS
15010 PRINT "TESTING LEVEL COMMAND":PRINT
15020 PRINT "ENTER OFFSET TO READ LEVEL(-2..+2):";:INPUT OFFSETZ

```

COMPILED BASIC TEST PROGRAM

```
15030 CMDERR%=SPACE$(200)
15040 CALL LEVEL(OFFSETZ,CMDERR%)
15050 PRINT "ERROR STRING:",CMDERR%
15060 PRINT "DO AGAIN? (Y/N):";:INPUT DOAGAIN%
15070 IF DOAGAIN% <> "N" THEN GOTO 15010
15080 RETURN
16000 REM*****
16005 REM SUBROUTINE TO TEST TRANSMIT COMMAND
16007 REM*****
16010 CLS
16020 PRINT "TESTING TRANSMIT COMMAND":PRINT
16030 PRINT "ENTER DESTINATION (0=PRINTER, 1= AUXPORT, 2=LED):";:INPUT DESTZ
16040 IF DESTZ=0 THEN PRINT "ENTER PRINT POSITION: ";:INPUT PRNTPOSZ
16050 IF DESTZ=1 THEN PRNTPOSZ=0
16060 IF DESTZ=2 THEN PRINT "ENTER VALUE (0..15) TO BE DISPLAYED: ";:INPUT PRNTPOSZ
16070 XMITSTR%="ABC"
16080 IF DESTZ = 0 OR DESTZ=1 THEN PRINT "ENTER STRING TO TRANSMIT: ";:INPUT XMITSTR%
16085 CMDERR%=SPACE$(200)
16090 CALL TRANSMIT(DESTZ,PRNTPOSZ,XMITSTR%,CMDERR%)
16100 PRINT "ERROR STRING:",CMDERR%
16110 PRINT "DO AGAIN? (Y/N):";:INPUT DOAGAIN%
16120 IF DOAGAIN% <> "N" THEN GOTO 16020
16130 RETURN
17000 REM*****
17005 REM SUBROUTINE TO TEST RECV COMMAND
17008 REM*****
17010 CLS
17020 PRINT "TESTING RECV COMMAND":PRINT
17030 PRINT "ENTER STRING AT AUX PORT TERMINAL AND PRESS"
17040 PRINT " RETURN AT TERMINAL TO TERMINATE OR ESCAPE ON PC TO TERMINATE"
17050 CMDERR%=SPACE$(200)
17055 RECVSTR% = SPACE$(254)
17060 CALL RECV(RECVSTR%,CMDERR%)
17080 PRINT "INPUT STRING: ";:PRINT RECVSTR%
17090 PRINT "ERROR STRING:",CMDERR%
17100 PRINT "DO AGAIN? (Y/N):";:INPUT DOAGAIN%
17110 IF DOAGAIN% <> "N" THEN GOTO 17020
17120 RETURN
18000 REM*****
18002 REM SUBROUTINE TO DISPLAY SHEET BUFFER OR SKUNK TABLE
18004 REM*****
18005 CLS
18010 PRINT "DISPLAY SHEET BUFFER OR SKUNK TABLE"
18020 PRINT
18030 PRINT "1. DISPLAY SHEET BUFFER"
18040 PRINT "2. DISPLAY SKUNK TABLE"
18050 PRINT "3. RETURN TO MAIN MENU"
18060 PRINT
18070 PRINT "ENTER SELECTION FROM ABOVE (1..3): ";
```



```

18080 INPUT KZ
18090 IF KZ = 3 THEN RETURN
18100 IF KZ < 1 OR KZ > 3 THEN GOTO 18020
18110 CLS
18120 IF KZ = 1 THEN GOSUB 19000 ELSE GOSUB 20000
18130 GOTO 18020
19000 REM*****
19002 REM   DISPLAY SHEET BUFFER
19004 REM*****
19005 CLS
19010 PRINT "DISPLAY SHEET BUFFER"
19020 PRINT
19030 PRINT "ENTER DESIRED ROW (1..99) OF SHEET TO BE DISPLAYED: ";
19040 INPUT STARTROWZ
19050 IF STARTROWZ < 1 OR STARTROWZ >99 THEN GOTO 19010
19052 CODESEGZ=0
19053 CALL GETSEG(CODESEGZ)
19055 DEF SEG =CODESEGZ
19060 SHTBUFZ= PEEK(16) + 256* PEEK(17) 'LOCATION OF POINTER TO SHEET BUFFER
19070 PRINT
19080 PRINT "ROW LOCATION      READ LEVELS"
19090 PRINT
19100 IF STARTROWZ > 89 THEN ENDROWZ=99 ELSE ENDROWZ = STARTROWZ + 10
19110 FOR KZ = STARTROWZ TO ENDROWZ
19120 PRINT USING "###";KZ;
19130 PRINT USING "   #####   ";(KZ-1)*48;
19140 FOR EACHMARKZ = 0 TO 47
19150 MLEVELZ= PEEK(SHTBUFZ+(KZ-1)*48 + EACHMARKZ) -48 'SUBTRACT OFF ASCII PREFIX
19160 PRINT USING "#"; MLEVELZ;
19170 NEXT EACHMARKZ
19180 PRINT
19190 NEXT KZ 'DO NEXT ROW
19200 PRINT "DO AGAIN ? (Y/N): "; :INPUT DOAGAIN$
19210 IF DOAGAIN$ <> "N" THEN GOTO 19010
19220 RETURN
20000 REM*****
20002 REM   DISPLAY SKUNK TABLE
20004 REM*****
20005 CLS
20010 PRINT "DISPLAY SKUNK TABLE "
20030 PRINT
20040 PRINT "ENTER DESIRED LOCATION IN TABLE(1..99): ";
20050 INPUT KZ
20060 IF KZ < 1 OR KZ > 99 THEN GOTO 20030
20062 CODESEGZ = 0
20063 CALL GETSEG(CODESEGZ) 'GET CODE SEGMENT OF ASSY MODULE
20065 DEF SEG =CODESEGZ
20070 SKTBLOCZ = PEEK(18) +256 *PEEK(19)

```

COMPILED BASIC TEST PROGRAM

```
20080 PRINT "ENTRY DOC# CELLS TRACKS NMARKS MARKS[1..47]"
20090 PRINT
20095 IF KZ >89 THEN LZ = 99 ELSE LZ = KZ + 10
20100 FOR MZ = KZ TO LZ
20110 PRINT USING " ### ";MZ;
20120 FOR NZ = 0 TO 3
20130 PRINT USING " ### "; PEEK(SKTBLOCZ + ((MZ-1)*10)+NZ);
20140 NEXT NZ
20150 PRINT " ";
20160 FOR NZ = 4 TO 9 : PRINT USING "###";PEEK(SKTBLOCZ+((MZ-1)*10)+NZ);
20170 NEXT NZ
20180 PRINT
20190 NEXT MZ
20200 PRINT
20210 PRINT "DO AGAIN ? (Y/N): ";INPUT DOAGAIN$
20220 IF DOAGAIN$ <> "N" THEN GOTO 20030
20230 RETURN
```

INTERPRETIVE BASIC TEST PROGRAM

```

50 REM  FILENAME: ITESTER   INTERPRETIVE BASIC TEST ROUTINE
55 REM  FILE CREATED: 12-JUL-83
60 REM  LAST REVISED: 12-JUL-83  10:00
65 REM  CONTEXT: TEST PROGRAM FOR INTERPRETIVE BASIC
67 REM                VERSION OF MICROSCANNER COMMANDS
70 REM
74 REM
75 REM  NOTE: THE LINES 100-190 ARE COMMENTED OUT WHEN USING COMPILE BASIC
77 REM
78 REM
100 DEF SEG=&H1CD4
110 BLOAD "B:comnds.bas",0
120 SCAN =PEEK(0) + 256 * PEEK(1)
130 GRID =PEEK(2) + 256 * PEEK(3)
140 SKUNK =PEEK(4) + 256 * PEEK(5)
150 CONTROL =PEEK(6) + 256 * PEEK(7)
160 SETUP  =PEEK(8) + 256 * PEEK(9)
170 LEVEL  =PEEK(10) + 256 * PEEK(11)
180 TRANSMIT=PEEK(12) + 256 * PEEK(13)
190 RECV   =PEEK(14) + 256 * PEEK(15)
200 DIM GRIDARGZ(8)
210 DIM MARKSZ(48)
220 REM*****
230 REM  MENU FOR TEST PROGRAM
240 REM*****
1000 PRINT "1. CONTROL COMMAND      4. SKUNK COMMAND      7. TRANSMIT COMMAND"
1010 PRINT "2. SETUP COMMAND        5. SCAN COMMAND      8. RECV COMMAND  "
1020 PRINT "3. GRID COMMAND             6. LEVEL COMMAND    "
1030 PRINT: PRINT "9. QUIT"
1040 PRINT: PRINT "ENTER SELECTION (1..9) FROM ABOVE: ";
1050 INPUT AZ
1055 IF AZ >9 THEN GOTO 1000
1060 IF AZ=9 THEN SYSTEM
1130 ON AZ GOSUB 10000,11000,12000,13000,14000,15000,16000,17000
1140 GOTO 1000
10000 REM*****
10005 REM SUBROUTINE TO TEST CONTROL COMMAND
10007 REM*****
10010 CLS
10020 PRINT "TESTING CONTROL COMMAND":PRINT
10030 PRINT "1. RELEASE DOCUMENT      3. SELECT AUX PORT"
10040 PRINT "2. STOP SCANNER          4. SELECT SCANNER"
10050 PRINT: PRINT "ENTER SELECTION FROM ABOVE (1..4)"
10060 INPUT CTRLPTZ
10070 CMDERR$=SPACE$(20)
10080 CALL CONTROL(CTRLPTZ,CMDERR$)
10090 PRINT "ERROR CODE: ";CMDERR$
10100 PRINT "DO AGAIN? (Y/N):";:INPUT DOAGAIN$

```

INTERPETIVE BASIC TEST PROGRAM

```
10110 IF DOAGAIN$ <> "N" THEN GOTO 10020
10120 RETURN
11000 REM*****
11005 REM  SUBROUTINE TO TEST SETUP COMMAND
11007 REM*****
11010 CLS
11020 PRINT "TESTING SETUP COMMAND":PRINT
11030 PRINT "ENTER BAUD RATE:";:INPUT BAUDZ
11040 PRINT "ENTER PARITY(O,E,N):";:INPUT PARITY$
11050 PRINT "ENTER DATA BITS:";:INPUT DATABITSZ
11060 PRINT "ENTER STOP BITS:";:INPUT STOPBITSZ
11070 PRINT "ENTER BOARD SELECT:";:INPUT PORTZ
11080 PARITYZ=ASC(PARITY$)
11085 CMDERR$=SPACE$(200)
11090 CALL SETUP(BAUDZ,PARITYZ,DATABITSZ,STOPBITSZ,PORTZ,CMDERR$)
11100 PRINT "ERROR STRING:",CMDERR$
11110 PRINT "DO AGAIN? (Y/N):";:INPUT DOAGAIN$
11120 IF DOAGAIN$ <> "N" THEN GOTO 11020
11130 RETURN
12000 REM*****
12005 REM  SUBROUTINE TO TEST GRID ROUTINE
12007 REM*****
12010 CLS
12015 PRINT "TESTING GRID COMMAND":PRINT
12020 PRINT "INPUT TYPE";:INPUT GRIDARGZ(0)
12030 PRINT "INPUT CLASS";:INPUT GRIDARGZ(1)
12040 PRINT "INPUT  SX";:INPUT GRIDARGZ(2)
12050 PRINT "INPUT  EX";:INPUT GRIDARGZ(3)
12060 PRINT "INPUT  IX";:INPUT GRIDARGZ(4)
12070 PRINT "INPUT  SY";:INPUT GRIDARGZ(5)
12080 PRINT "INPUT  EY";:INPUT GRIDARGZ(6)
12090 PRINT "INPUT  IY";:INPUT GRIDARGZ(7)
12100 Y#=FRE(0)
12110 EDSTATZ=99
12120 GRIDSTR$=SPACE$(254):CMDERR$=SPACE$(200)
12130 ARGPTRZ= VARPTR(GRIDARGZ(0))
12140 CALL GRID(ARGPTRZ,GRIDSTR$,EDSTATZ,CMDERR$)
12150 PRINT "GRID STRING:",GRIDSTR$ :PRINT "EDIT STATUS:",EDSTATZ
12160 PRINT "ERROR STRING:",CMDERR$
12180 PRINT "DO AGAIN? (Y/N):";:INPUT DOAGAIN$
12190 IF DOAGAIN$ <> "N" THEN GOTO 12020
12200 RETURN
13000 REM*****
13005 REM  SUBROUTINE TO TEST SKUNK COMMAND
13007 REM*****
13010 CLS
13020 PRINT "TESTING SKUNK COMMAND":PRINT
13030 PRINT "INPUT  DOC#:";:INPUT DOCNUMZ
13040 PRINT "INPUT  CELLS:";:INPUT CELLSZ
```

```

13050 PRINT "INPUT TRACKS:";:INPUT TRACKSZ
13060 PRINT "INPUT NMARKS:";:INPUT NUMMARKSZ
13070 FOR K=0 TO 50 : PRINT "INPUT MARK LOCATION,99 TO EXIT:";
13075 INPUT MARKSZ(K)
13080 IF MARKSZ(K)=99 THEN GOTO 13100
13090 NEXT K
13100 CMDERR$=SPACE$(200)
13110 Y$=FRE(0)
13120 MARKPTRZ=VARPTR(MARKSZ(0))
13130 CALL SKUNK(DOCNUMZ,CELLSZ,TRACKSZ,NUMMARKSZ,MARKPTRZ,CMDERR$)
13140 PRINT "ERROR STRING:";CMDERR$
13150 PRINT "DO AGAIN? (Y/N):";:INPUT DOAGAIN$
13160 IF DOAGAIN$ <> "N" THEN GOTO 13020
13170 RETURN
14000 REM*****
14005 REM SUBROUTINE TO TEST SCAN ROUTINE
14007 REM*****
14010 CLS
14020 PRINT "TESTING SCAN COMMAND":PRINT
14030 PRINT "ENTER 2 FOR SCAN, 3 FOR RE-TRANSMIT:";:INPUT READTYPEZ
14040 CMDERR$=SPACE$(200):DOCZ=0
14050 CALL SCAN(DOCZ,READTYPEZ,CMDERR$)
14060 PRINT "I.D. NUMBER OF DOCUMENT SCANNED: ";
14065 PRINT USING "##"; DOCZ
14070 PRINT "ERROR STRING:";CMDERR$
14080 PRINT "DO AGAIN? (Y/N):";:INPUT DOAGAIN$
14090 IF DOAGAIN$ <> "N" THEN GOTO 14020
14100 RETURN
15000 REM*****
15005 REM SUBROUTINE TO TEST LEVEL COMMAND
15007 REM*****
15008 CLS
15010 PRINT "TESTING LEVEL COMMAND":PRINT
15020 PRINT "ENTER OFFSET TO READ LEVEL(-2..+2):";:INPUT OFFSETZ
15030 CMDERR$=SPACE$(200)
15040 CALL LEVEL(OFFSETZ,CMDERR$)
15050 PRINT "ERROR STRING:";CMDERR$
15060 PRINT "DO AGAIN? (Y/N):";:INPUT DOAGAIN$
15070 IF DOAGAIN$ <> "N" THEN GOTO 15010
15080 RETURN
16000 REM*****
16005 REM SUBROUTINE TO TEST TRANSMIT COMMAND
16007 REM*****
16010 CLS
16020 PRINT "TESTING TRANSMIT COMMAND":PRINT
16030 PRINT "ENTER DESTINATION (0=PRINTER, 1= AUXPORT, 2=LED):";:INPUT DESTZ
16040 IF DESTZ=0 THEN PRINT "ENTER PRINT POSITION:";:INPUT PRNTPOSZ

```

INTERPETIVE BASIC TEST PROGRAM

```
16050 IF DESTZ=1 THEN PRINTPOSZ=0
16060 IF DESTZ=2 THEN PRINT "ENTER VALUE (0..15) TO BE DISPLAYED:";:INPUT PRNTPOSZ
16070 XMITSTR$="ABC"
16080 IF DESTZ = 0 OR DESTZ=1 THEN PRINT "ENTER STRING TO TRANSMIT:";:INPUT XMITSTR$
16085 CMDERR$=SPACE$(200)
16090 CALL TRANSMIT(DESTZ,PRNTPOSZ,XMITSTR$,CMDERR$)
16100 PRINT "ERROR STRING:",CMDERR$
16110 PRINT "DO AGAIN? (Y/N):";:INPUT DOAGAIN$
16120 IF DOAGAIN$ <> "N" THEN GOTO 16020
16130 RETURN
17000 REM:*****
17005 REM SUBROUTINE TO TEST RECV COMMAND
17008 REM:*****
17010 CLS
17020 PRINT "TESTING RECV COMMAND":PRINT
17030 PRINT "ENTER STRING AT AUX PORT TERMINAL AND PRESS"
17040 PRINT "  RETURN AT TERMINAL TO TERMINATE OR ESCAPE ON PC TO TERMINATE"
17050 CMDERR$=SPACE$(200)
17055 RECVSTR$ = SPACE$(254)
17060 CALL RECV(RECVSTR$,CMDERR$)
17080 PRINT "INPUT STRING:";:PRINT RECVSTR$
17090 PRINT "ERROR STRING:",CMDERR$
17100 PRINT "DO AGAIN? (Y/N):";:INPUT DOAGAIN$
17110 IF DOAGAIN$ <> "N" THEN GOTO 17020
17120 RETURN
```

```

(FILENAME: PTESTER   PASCAL MICROSCANNER COMMANDS TEST PROGRAM
FILE CREATED: 20-JUN-83
LAST REVISED: 15-JUL-83 12:30
CONTEXT      : IBM-PASCAL IMPLEMENTATION OF MICROSCANNER COMMANDS
              EXAMPLE TEST PROGRAM
}

```

```
PROGRAM PTESTER(INPUT, OUTPUT);
```

```
(**INCLUDE:'SCANDECL.PAS'*)
```

```

      (THIS FILE HAS ALL THE SPECIAL TYPE DECLARATIONS
      AND EXTERNAL DECLARATIONS FOR MICROSCANNER COMMANDS
      AND MUST BE INCLUDED INTO ANY PROGRAM WHICH USES
      MICROSCANNER COMMANDS. )

```

```
VAR
```

```

  ACHAR : CHAR;
  AGAIN : BOOLEAN;
  K,MENUSELECT : INTEGER;
  CMDERR,GRIDSTR : MSCSTR;
  PROCEDURE DOAGAIN(VAR DUM1 : BOOLEAN);

```

```
{*****}
```

```

BEGIN;
  WRITELN;
  WRITELN(OUTPUT, 'ERROR STRING: ',CMDERR);
  WRITELN;
  WRITE(OUTPUT, 'DO AGAIN? (Y/N): ');
  READLN(INPUT, ACHAR);
  WRITELN;
  IF ACHAR = CHR('N') THEN DUM1 := FALSE
  ELSE DUM1 := TRUE;
END; {OF PROC DOAGAIN}

```

```
{*****}
```

```
PROCEDURE TCONTROL; {TEST CONTROL COMMAND}
```

```
{*****}
```

```
VAR
```

```
  CTRLOPT:INTEGER;
```

```
BEGIN;
```

```

  REPEAT
    COPYLST('DUMMY STRING',CMDERR);
    WRITELN;
    WRITELN;

```

PASCAL TEST PROGRAM

```
WRITELN('TESTING CONTROL COMMAND');
WRITELN;
WRITELN('1. RELEASE DOCUMENT      3. SELECT AUX PORT');
WRITELN('2. STOP SCANNER          4. SELECT SCANNER');
WRITELN;
WRITE('ENTER SELECTION FROM ABOVE (1..4): ');
READLN(INPUT, CTRLPT);

CONTROL( CTRLPT, CMDERR); (MICROSCANNER COMMAND CALL)

DOAGAIN(AGAIN);
UNTIL NOT AGAIN;
END; (OF PROC TCONTROL)

{*****}
PROCEDURE TSETUP;

{*****}
VAR
    BAUD,PARITY,DATABITS,STOPBITS,PORTSEL : INTEGER;

BEGIN;
REPEAT
    COPYLST('DUMMY STRING',CMDERR);
    WRITELN;
    WRITELN;
    WRITELN(' TESTING SET-UP COMMAND');
    WRITELN;
    WRITE('ENTER BAUD RATE: ');
    READLN( INPUT, BAUD);
    WRITE('ENTER PARITY ("O", "E", "N"): ');
    READLN(INPUT,ACHAR);
    PARITY:= ORD(ACHAR);
    WRITE('ENTER DATA BITS: ');
    READLN(INPUT,DATABITS);
    WRITE('ENTER STOP BITS: ');
    READLN(INPUT,STOPBITS);
    WRITE('ENTER BOARD SELECT: ');
    READLN(INPUT,PORTSEL);

    SETUP( BAUD, PARITY, DATABITS, STOPBITS, PORTSEL, CMDERR);

DOAGAIN(AGAIN);
UNTIL NOT AGAIN;
END; (OF PROC TSETUP)
```



```
(*****)  
PROCEDURE TGRID;
```

```
(*****)  
VAR  
  GRIDARG : GRIDARA;  
  EDSTAT : INTEGER;
```

```
BEGIN;  
  REPEAT  
    COPYLST('DUMMY STRING',GRIDSTR);  
    COPYLST('DUMMY STRING',CMDERR);  
    WRITELN;  
    WRITELN;  
    WRITELN('TESTING GRID COMMAND');  
    WRITELN;  
    WRITE('ENTER TYPE: ');  
    READLN(GRIDARG[0]);  
    WRITE('ENTER CLASS: ');  
    READLN(GRIDARG[1]);  
    WRITE('ENTER SX: ');  
    READLN(GRIDARG[2]);  
    WRITE('ENTER EX: ');  
    READLN(GRIDARG[3]);  
    WRITE('ENTER IX: ');  
    READLN(GRIDARG[4]);  
    WRITE('ENTER SY: ');  
    READLN(GRIDARG[5]);  
    WRITE('ENTER EY: ');  
    READLN(GRIDARG[6]);  
    WRITE('ENTER IY: ');  
    READLN(GRIDARG[7]);  
  
    GRID(GRIDARG, GRIDSTR, EDSTAT, CMDERR);  
  
    WRITELN;  
    WRITELN('EDIT STATUS: ', EDSTAT);  
    WRITE('GRID STRING:', GRIDSTR);  
    WRITELN;  
    DOAGAIN(AGAIN);  
  UNTIL NOT AGAIN;  
END; (OF PROC TGRID)
```

PASCAL TEST PROGRAM

```
(*****)  
PROCEDURE TSKUNK; {TEST ROUTINE FOR THE SKUNK COMMAND}
```

```
(*****)
```

```
VAR
```

```
  L, DOCNUM, CELLS, TRACKS, NUMMARKS : INTEGER;  
  MARKS : SKARAY;
```

```
BEGIN;
```

```
  REPEAT
```

```
    COPYLST('DUMMY STRING',CMDERR);  
    WRITELN;  
    WRITELN;  
    WRITELN('TESTING SKUNK COMMAND');  
    WRITELN;  
    WRITE('ENTER DOCUMENT NUMBER: ');  
    READLN(DOCNUM);  
    WRITE('ENTER NUMBER OF CELLS: ');  
    READLN(CELLS);  
    WRITE('ENTER NUMBER OF TRACKS: ');  
    READLN(TRACKS);  
    WRITE('ENTER # OF SKUNK MARKS: ');  
    READLN(NUMMARKS);
```

```
  L := 0;
```

```
  K := 0;
```

```
  WHILE ((L <> 99) AND (K < 47)) DO
```

```
    BEGIN;
```

```
      WRITE('INPUT MARK LOCATION, 99 TO EXIT: ');
```

```
      READLN(L);
```

```
      MARKS[K] := L;
```

```
      K := K+1;
```

```
    END; {OF WHILE}
```

```
  SKUNK(DOCNUM, CELLS, TRACKS, NUMMARKS, MARKS, CMDERR);
```

```
  DOAGAIN(AGAIN);
```

```
  UNTIL NOT AGAIN;
```

```
END; {OF PROC TSKUNK}
```

```

{*****}
PROCEDURE TSCAN;

```

```

{*****}
VAR
  DOC, READTYPE : INTEGER;

```

```

BEGIN;
  REPEAT
    COPYLST('DUMMY STRING',CMDERR);
    WRITELN;
    WRITELN;
    WRITELN('TESTING SCAN COMMAND');
    WRITELN;
    WRITE('ENTER 2 FOR SCAN, 3 FOR RE-TRANSMIT: ');
    READLN(READTYPE);

    SCAN( DOC, READTYPE, CMDERR);

    WRITE('ASSIGNED NUMBER OF DOCUMENT SCANNED: ');
    WRITELN(DOC);
    DDAGAIN(AGAIN);
  UNTIL NOT AGAIN;
END; (OF PROC TSCAN)

```

```

{*****}
PROCEDURE TLEVEL;

```

```

{*****}
VAR
  OFFSET: INTEGER;

```

```

BEGIN;
  REPEAT
    COPYLST('DUMMY STRING',CMDERR);
    WRITELN;
    WRITELN;
    WRITELN('TESTING LEVEL COMMAND');
    WRITELN;
    WRITE('ENTER OFFSET TO READ LEVEL (-2..+2): ');
    READLN(OFFSET);
    LEVEL( OFFSET, CMDERR);

    DDAGAIN(AGAIN);
  UNTIL NOT AGAIN;
END; (OF PROC TLEVEL)

```

PASCAL TEST PROGRAM

```
{*****}
PROCEDURE TXMIT;
```

```
{*****}
```

```
TYPE ONESTR = STRING(1);
VAR
  PRNTPOS, DEST : INTEGER;
  XMITSTR : MSCSTR;
  F,G : FILE OF CHAR;
  NEXTCHAR: ONESTR;
  MY_EDLN : BOOLEAN;
```

```
{THE PROCEDURE 'OPEN CONSOLE, AND FUNCTION INKEY ARE NECESSARY BECAUSE THE
READLN PROCEDURE WILL ONLY INPUT 126 CHARACTERS AND WE NEED TO INPUT
254 CHARACTERS TO TEST THE XMIT COMMAND.
}
```

```
PROCEDURE OPEN_CONSOLE;
BEGIN;
  ASSIGN(F,'USER');
  RESET(F);
  ASSIGN(G,'USER');
  REWRITE(G);
END; {PROCEDURE OPEN_CONSOLE}
```

```
FUNCTION INKEY:ONESTR;
VAR
  INCHAR : CHAR;
  TMPSTR :ONESTR;

BEGIN;
  REPEAT GET(F) UNTIL F^ <> CHR(0);
  INCHAR := F^;
  WRITE(G,INCHAR);
  TMPSTR[1] := INCHAR;
  INKEY := TMPSTR;
END; {OF FUNCTION INKEY}
```

```
BEGIN; {OF PROCEDURE TXMIT}
  OPEN_CONSOLE;
  REPEAT
    COPYLST('DUMMY STRING',XMITSTR);
    COPYLST('DUMMY STRING',CMDERR);
    WRITELN;
    WRITELN;
    WRITELN('TESTING TRANSMIT COMMAND');
    WRITELN;
    WRITE('ENTER DESTINATION ( 0 = PRINTER, 1 = AUX PORT, 2 = LED): ');
```

```
READLN(DEST);
CASE DEST OF
0:
  BEGIN;
  WRITE('ENTER PRINT POSITION: ');
  READLN(PRNTPOS);
  WRITE('ENTER STRING TO BE PRINTED: ');
  READLN(XMITSTR);
  END;
1:
  BEGIN;
  WRITE('ENTER STRING TO BE SENT TO AUX PORT: ');
  K:= 0;
  MY_EOLN := FALSE;
  XMITSTR.LEN := 0; (CLEAR XMITSTR)
  REPEAT
    NEXTCHAR := INKEY;
    IF NEXTCHAR[1] <> CHR(13) THEN
      BEGIN;
        CONCAT(XMITSTR,NEXTCHAR);
        MY_EOLN := FALSE;
      END
    ELSE MY_EOLN := TRUE;

    K := K+1;
  UNTIL (K = 254) OR (MY_EOLN =TRUE);
  WRITELN;
  PRNTPOS := 0;
  END;
2:
  BEGIN;
  WRITE('ENTER VALUE (0..15) TO BE DISPLAYED: ');
  READLN(PRNTPOS);
  END;
3..MAXINT: PRNTPOS :=0;

END; (OF CASE)

TRANSMIT( DEST, PRNTPOS, XMITSTR, CMDERR);

DOAGAIN(AGAIN);
UNTIL NOT AGAIN;
END; (OF TXMIT COMMAND )
```

PASCAL TEST PROGRAM

```
{.....}
PROCEDURE TREC;V;
```

```
{.....}
VAR
  INSTRING : MSCSTR;
```

```
BEGIN;
  REPEAT
    COPYLST('DUMMY STRING',INSTRING);
    COPYLST('DUMMY STRING',CMDERR);
    WRITELN;
    WRITELN;
    WRITELN('TESTING THE RECEIVE COMMAND');
    WRITELN;
    WRITELN('ENTER STRING AT AUX PORT TERMINAL AND PRESS RETURN ');
    WRITELN(' AT TERMINAL OR ESCAPE ON PC TO TERMINATE RECV COMMAND');
    WRITELN;

    RECV( INSTRING, CMDERR);

    WRITELN('INPUT STRING:', INSTRING);
    DOAGAIN(AGAIN);
  UNTIL NOT AGAIN;
END; {OF PROC TREC}
```

```
{.....}
BEGIN; {MAIN PROGRAM PTESTER}
```

```
{.....}
  REPEAT
    WRITELN;
    WRITELN;
    WRITELN('1. CONTROL COMMAND      4. SKUNK COMMAND      7. TRANSMIT COMMAND');
    WRITELN('2. SETUP COMMAND      5. SCAN COMMAND      8. RECV COMMAND');
    WRITELN('3. GRID COMMAND      6. LEVEL COMMAND');
    WRITELN;
    WRITELN('9. QUIT ');
    WRITELN;
    WRITE('ENTER SELECTION (1..9) FROM ABOVE: ');
    READLN(MENUSELECT);
```

```
CASE MENUSELECT OF
1: TCONTROL;
2: TSETUP;
3: TGRID;
4: TSKUNK;
5: TSCAN;
6: TLEVEL;
7: TXMIT;
8: TREC;
9:
END; (OF CASE)
UNTIL MENUSELECT = 9;
END. (OF MAIN PROGRAM)
```



APPENDIX C

PASCAL SCANDECL.PAS FILE

OVERVIEW: SCANDECL.PAS FILE

In Section Four, under program structure, programmers are advised to insert the line (*\$INCLUDE:'SCANDECL.PAS'*) into their programs immediately following the program heading. That line causes the file SCANDECL.PAS to be included in the application program. The file contains external type declarations necessary for using Micro Scanner Commands.

EXPLANATION: SCANDECL.PAS FILE

The contents of the file are listed below.

```
( FILENAME:    SCANDECL.PAS
  FILE CREATED: 12-JUL-83
  LAST REVISED: 12-JUL-83 18:00
  CONTEXT:     IBM PASCAL IMPLEMENTATION OF MICROSCANNER COMMANDS
```

```
THIS FILE HAS THE TYPE AND EXTERNAL DECLARATIONS REQUIRED FOR
MICROSCANNER COMMANDS. IT S CONTENTS MUST BE INCLUDED IN A PROGRAM
WHICH USES MICRO SCANNER COMMANDS. THIS IS EASILY ACCOMPLISHED WITH
THE FOLLOWING LINE IN THE APPLICATION PROGRAM:
  $INCLUDE:SCANDECL.PAS.
THIS "INCLUDE" STATEMENT SHOULD FOLLOW DIRECTLY AFTER THE PROGRAM HEADING.
}
```

TYPE

```
MSCSTR = LSTRING(254);
GRIDARA= ARRAY [0..7] OF INTEGER;
SKARAY = ARRAY [0..47] OF INTEGER;
```

```
{*****}
(      EXTERNALLY REFERENCED MICROSCANNER COMMAND DECLARATIONS      )
{*****}
```

```
PROCEDURE CONTROL(CONST CTRLOPT:INTEGER; VAR CMDERR:MSCSTR);
  EXTERNAL;
```

```
PROCEDURE SETUP(CONST BAUD:INTEGER; CONST PARITY:INTEGER; CONST DATABITS:INTEGER;
  CONST STOPBITS:INTEGER; CONST PORTSEL:INTEGER;
  VAR CMDERR:MSCSTR);
  EXTERNAL;
```

```
PROCEDURE GRID(CONST GRIDARG:GRIDARA; VAR GRIDSTR:MSCSTR; VAR EDSTAT:INTEGER;
  VAR CMDERR:MSCSTR);
  EXTERNAL;
```

PASCAL SCANDECL.PAS FILE

EXPLANATION: SCANDECL.PAS FILE (cont.)

PROCEDURE SKUNK(CONST DOCNUM:INTEGER; CONST CELLS:INTEGER; CONST TRACKS:INTEGER;
CONST NUMMARKS:INTEGER;
CONST MARKS:SKARAY; VAR CMDERR:MSCSTR);

EXTERNAL;

PROCEDURE SCAN(VAR DOC:INTEGER; CONST READTYPE:INTEGER;
VAR CMDERR:MSCSTR);

EXTERNAL;

PROCEDURE LEVEL(CONST OFFSET:INTEGER; VAR CMDERR:MSCSTR);

EXTERNAL;

PROCEDURE TRANSMIT(CONST DEST :INTEGER; CONST PRNTPOS:INTEGER;
CONST XMITSTR:MSCSTR; VAR CMDERR:MSCSTR);

EXTERNAL;

PROCEDURE RECV(VAR INSTRING:MSCSTR; VAR CMDERR:MSCSTR);

EXTERNAL;

APPENDIX D

LINKING COMMANDS TO
APPLICATION PROGRAM

Introduction.....D-2
Linking.....D-3
Diskette Contents.....D-5

INTRODUCTION

In order to utilize micro scanner commands in application programs, they must be linked together. This appendix describes the linking process.

OVERVIEW: LINKING

The linking requirements for each language are described below. For detailed descriptions of the linking process, refer to the IBM - Personal Computer Language Series - BASIC Compiler and Pascal Compiler.

EXPLANATION: COMPILED BASIC

The accompanying examples explain how to link the commands to the application program. The small letters represent prompts generated by the microcomputer; the capital letters represent responses typed by the programmer.

The first example is for running an application program which is compiled with the "/O" option. The second example is for an application program compiled without the "/O" option.

The linking procedure is started by typing in the word LINK. Then the object module is requested. The object module refers to the program files which will be connected to the commands. Type in the name of the application program and CBSCAN. The run file line states that the file is now executable. The list file line allows the programmer to request a map. Type in BASCOM after "Libraries:[.lib]:".

Linking for a program compiled without the "/O" option works virtually the same except the last line. Type in BASRUN after "libraries:[.lib]:".

LINKING - COMPILED BASIC
/O OPTION

c>LINK

IBM Personal Computer Linker
Version 1.10 (C)Copyright IBM Corp.
1982

Object Modules [.obj]: FILENAME
CBSCAN

Run File [filename.exe]:
List File [nul.map]:
Libraries [.lib]: BASCOM

LINKING COMPILED BASIC
NO /O OPTION

C>LINK

IBM Personal Computer Linker
Version 1.10 (C)Copyright IBM Corp.
1982

Object Modules [.obj]: FILENAME
CBSCAN

Run File [filename.exe]:
List File [nul.map]:
Libraries [.lib]: BASRUN

LINKING

LINKING - PASCAL

C>LINK

IBM Personal Computer Linker
Version 1.10 (C)Copyright IBM Corp.
1982

Object Modules [.obj]: FILENAME
PSCAN

Run File [filename.exe]:

List File [nul.map]:

Libraries [.lib]: PASCAL

EXPLANATION: PASCAL

Linking in Pascal is similar to linking in Compiled Basic. However, the file name is PSCAN. Also, after "Libraries:[.lib]:", enter PASCAL.

EXPLANATION: INTERPRETIVE BASIC

No separate linking process is required. The commands were already linked to the application program in the loading process.

OVERVIEW: DISKETTE CONTENTS

The contents of the Micro Scanner Commands diskette are described below:

●Compiled Basic Files

CBSCAN.OBJ--Object file containing Compiled Basic implementation of microscanner commands. This file is linked to a Compiled Basic application program.

CTESTER.BAS--Source file containing the Compiled Basic test program.

CTESTER.EXE--Executable file containing the CTESTER program and microscanner commands.

LNKBAS.BAT--A sample batch file for linking a Compiled Basic application program to CBSCAN.OBJ.

●Interpretive Basic Files

COMNDS.BAS--The file containing microscanner commands which is loaded from within an Interpretive Basic application program. (With a 'BLOAD "COMNDS.BAS",0 ' command).

IBSCAN.OBJ--The object file containing unlinked and unlocated object code of the Interpretive Basic microscanner commands. This will allow the user to link his assembly language routine with microscanner commands to create a single ".EXE" file.

ITESTER.BAS--Interpretive Basic test program.

LNKIBSCN.BAT--Batch file to link IBSCAN.OBJ into IBSCAN.EXE.

●Pascal Files

PSCAN.OBJ--Object file containing the Pascal implementation of microscanner commands. This file is linked to a Compiled Pascal application program.

SCANDECL.PAS--Pascal source file meant to be "included" in the user's application program. It contains all the Pascal declarations required to interface to microscanner commands.

PTESTER.PAS--Source file containing Pascal test program.

DISKETTE CONTENTS

●Pascal Files (cont.)

PTESTER.EXE--Executable file containing the PTESTER test program and Pascal implementation of microscanner commands.

PASLNK.BAT--Batch file to link PSCAN.OBJ to a Pascal application program.

COMMENT SHEET

**Micro Scanner Commands User's Guide
For the Sentry PLUS™ System
Publication Number : 202 151 957**

FROM:

NAME:

BUSINESS ADDRESS:

COMMENTS:

**(Describe errors, suggested additions, or deletions,
including page numbers.)**

**NO POSTAGE STAMP NECESSARY IF MAILED IN U.S.A.
FOLD ON DOTTED LINES AND STAPLE**

FOLD



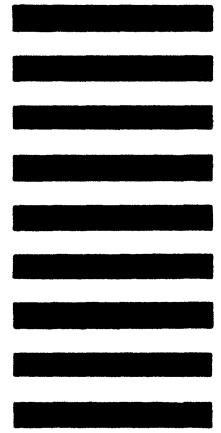
NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 1840 MINNEAPOLIS, MINN.

POSTAGE WILL BE PAID BY ADDRESSEE

National Computer Systems
c/o Publications
4401 West 76th Street
P.O. Box 9365
Minneapolis, Minnesota 55440



FOLD