

SOFTWARE DESIGN DOCUMENT
FOR FIELD CHECKING
SYSTEM

Submitted to: Geological Survey
of Canada

SKL Document #2100-12-002-01.0
Copy #3 15 December 1987

This document was produced
by scanning the original publication.

Ce document est le produit d'une
numérisation par balayage
de la publication originale.



Software Kinetics

SOFTWARE
DESIGN DOCUMENT
FOR
FIELD CHECKING SYSTEM

Submitted to: Geological Survey of Canada
Experimental Airborne Operations

SKL Document #2100-12-002

15 December 1987

Prepared by:

Wayne A. Reed
Wayne A. Reed

11-Dec-87
Date

Approved by:

T. M. Symchych
T. M. Symchych

Dec 11/87
Date



Software Kinetics

TABLE OF CONTENTS

Part I

	OVERVIEW
Section 1	DESIGN DESCRIPTION
Section 2	DEPENDENCY DIAGRAMS

Part II

Section 3	PROGRAM STRUCTURE
Appendix A	CONSTANTS
Appendix B	PROMPT STRINGS
Appendix C	STATUS/ERROR MESSAGES
Appendix D	HARDWARE CONFIGURATIONS



OVERVIEW

This document specifies the software design of the Field Checking System for the Experimental Airborne Operations of Geological Survey of Canada. The System will reside in the field laboratory during field operations.

Using a Beachcraft B80 Queenaire aircraft, sampling lines are flown to record aeromagnetic data using the Aeromagnetic Data Acquisition System, built internally at Energy Mines and Resources. This combined magnetic and navigational data is stored, during flight, on a rigid disk drive.

Magnetic data is also collected at a Diurnal Ground Station and stored on a hard disk. The hard disks can be removed from their acquisition systems and taken to the field laboratory for data verification and transfer to magnetic tape, a permanent storage medium. An IBM-AT microcomputer will be used to verify the recorded information, to plot geomagnetic fields on a printer/plotter, to perform fourth difference calculations and to copy data between hard disk and magnetic tape. The System will also allow the creation of an edited tape, to be sent to the Booth Street Laboratories of E.M.R. for computer compilation of the resultant aeromagnetic maps.



SECTION 1

DESIGN DESCRIPTION

	page
1.0 SCREEN PRESENTATION	1
1.1 Screen Layout	1
1.1.1 Presentation Area	1
1.1.2 Error/Status Line	4
1.1.3 Prompt Line	4
1.1.4 Input Line	4
1.2 Using The Screen Package	4
2.0 DATA STRUCTURES	6
2.1 Displays	6
2.1.1 Files	6
2.1.1.1 Template File	6
2.1.1.2 Value File	7
2.1.1.3 File Names	10
2.1.2 Display Windows	11
2.1.3 Presentation Area Field Values	12
2.2 Error Table	15
2.3 Prompt Table	18
2.4 Logical Unit Numbers	19
2.5 Recording Parameter Specifications	22
2.6 Airborne/Diurnal Data Sets and Header	25
2.7 Storage Device Specifications	27
2.8 Plot Table	28
2.9 Stacked Profile Table	29
2.10 Gradient Parameter Creation Table	39
2.11 Edit Table	41
3.0 FULL SCREEN EDITING	46
3.1 Cursor Movement	46
3.2 Field Modification	50
4.0 SYSTEM SPECIFICATIONS	52
4.1 Logical Unit Numbers	52
4.2 Recording Parameters	53
4.3 Airborne Data Character Set & Header	55
4.4 Diurnal Data Character Set & Header	57
4.5 Storage Device Specifications	58



5.0	MEDIUM CONTROL	59
5.1	Tape Control Functions	59
5.2	Disk Control Functions	60
5.3	Search/Copy Functions	60
6.0	BLOCK PRINTING	62
7.0	PLOTTING	65
7.1	Plot Specifications	65
7.1.1	Standard Plot	65
7.1.2	Successive Difference Plot	67
7.2	Plotting Process	68
8.0	STACKED PROFILES	70
8.1	Data Extraction	70
8.2	Inclusion Process	71
8.3	Plot Profiling Process	72
8.3.1	Free Format Profile	73
8.3.2	Time-Adjusted Profile	74
8.3.3	Other Format Profile	74
8.4	Exiting and Saving Stacked Profile Data	75
9.0	GRADIENT PARAMETER CREATION	76
10.0	EDIT FUNCTIONS	78
10.1	Batch Editor	78
10.1.1	Start/Stop Edit Limits	78
10.1.2	Output NNN Blocks	79
10.1.3	Add/Subtract a Value to a Parameter	79
10.1.4	Replace a Parameter Value	80
10.1.5	Delete/Insert a Parameter	81
10.1.5.1	Deletion	81
10.1.5.2	Insertion	81
10.1.6	Change One Digit	82
10.1.7	Check For Spike	83
10.1.8	Perform Edit Changes	84
10.2	Screen Editor	86
10.2.1	Reading a Block	86
10.2.2	Paging	86
10.2.2.1	Up	86
10.2.2.2	Down	87
10.2.3	Cursoring	87
10.2.4	Destructive Changes	88
10.2.5	Insertion	88
10.2.6	Deletion	89
10.2.7	Backward One Block	89
10.2.8	Error Checking	89
10.2.9	Writing a Data Block	90
10.2.10	Exiting Editor	90



SECTION 1

DESIGN DESCRIPTION

1.0 SCREEN PRESENTATION

The Field Checking System will employ menus to guide the Operator through the various options of the System. Screen presentations will also be used to set up specifications and data for certain options. Specialized software to perform these functions is available with the VENIX/86 Operating System. Referred to as the "Screen Package" this software can be thought of as three parts: (1) Screen updating; (2) Screen updating with user input; and (3) Cursor motion optimization. All three of these functions will be utilized in the Field Checking System Software.

1.1 Screen Layout

1.1.1 Presentation Area

A typical display will be broken up into four parts as shown in Figure 1-1-1. The largest portion will be known as the Presentation Area. Located between lines 0 through 21 inclusive and columns 0 through 79 inclusive, this zone will be outlined by a 'dashed box' with corner coordinates (containing a '+' character) as follows:

```
upper left  = (0,0)
upper right = (0,79)
lower left  = (21,0)
lower right = (21,79)
```



Within the Presentation Area the various menus will be displayed. This area will also be used by the Operator to enter data for some options. The cursor keys will provide the means for movement within this zone. In most cases these keys will be disabled when the display is presenting a menu: a message will inform the Operator of those instances when this is not true. For those displays requiring input from the Operator within the Presentation Area the cursor keys will be enabled.



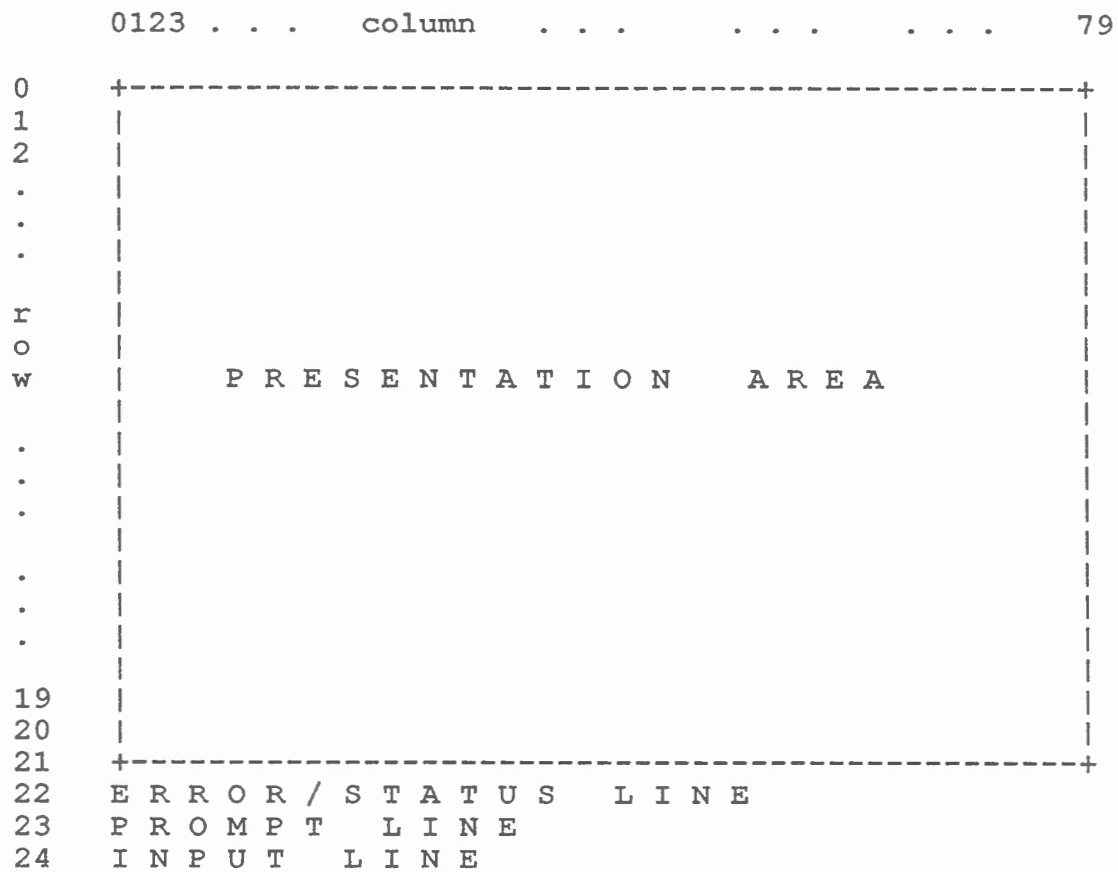


Figure 1-1-1 Screen Layout



1.1.2 Error/Status Line

Immediately below the Presentation Area will be the Error/Status Line. This will be located on line 22 of the Monitor, running the entire horizontal length of the screen. On this line system messages and error diagnostics will be displayed in inverse video. For the most part, this area of the screen will be referred to as the Error Line throughout this document.

1.1.3 Prompt Line

Below the Error Line will be the Prompt Line, line 23 of the monitor, which will also be eighty characters in length. All prompts will be issued on this line. The Operator will have no access to this area nor to the Error Line. When data entry is not done in the Presentation Area it will be made in the fourth and final zone of the display.

1.1.4 Input Line

Line 24 of the monitor will be the Input Line. The Operator will use the keyboard, when the cursor is present in this zone and so prompted by a message on the Prompt Line, to enter a response to an issued prompt.

1.2 Using The Screen Package

To incorporate the screen package into the Field Checking System the library routines of the package must be included in the software during compilation. To obtain the necessary types and variables defined, a line at the top of any software module referencing screen package functions must be added as follows:



2100-12-002.01.0

```
#include <curses.h>
```

When compiling, the library routines must be linked using a form as follows:

```
cc filename -lcurses -ltermplib
```



2.0 DATA STRUCTURES

2.1 Displays

A Display refers to any image presented on the IBM-AT Monitor. Menus will form the greatest number of images. Each display will consist of two components: (1) a 'template' which will contain the static text, etc. of the image and, (2) the 'values' which will be needed for presentation.

Both components will be saved as files on the IBM-AT Winchester Disk. When a particular display is to be presented the elements of the file will be combined to form the requested image. The following subsections describe the contents of these files.

2.1.1 Files

2.1.1.1 Template File

Each Template file will contain the following information:

- (1) the prompt to be issued when the display is first brought up on the monitor,
- (2) the (y,x) coordinate for the text,
- (3) the text string to be displayed at this coordinate.

The prompt will be the string to be presented on the Prompt line when the display first appears. The template file will contain a number which will represent the prompt. This three digit number will be used to index an internally-stored table containing the text of the limited number of prompts used by the Field Checking System.



#2100-12-002.01.0

The prompt number will be the first item of the template file. If no prompt is required for the display, a zero (000) will be placed in the template file to indicate that the Prompt line is to be left blank.

The (y,x) coordinate is the character position on the IBM-AT screen with (0,0) representing the upper, left corner. Y represents the column, and X represents the row position. Each coordinate value will be three characters in length.

The text will be exactly that to be presented: both lower-case and upper-case characters will be allowed along with any other printable characters. The text string delimiters will be double quotes (").

NOTE: Coordinates and text will be separated from one another by at least one blank space.

Figure 2-1-1-1 shows an example layout of a template file.

```
004
002 014 "T E S T   P A T T E R N"
004 010 " !#$%&' () *+,-./0123456789:;<=>?"
005 010 "ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^`"
006 010 "abcdefghijklmnopqrstuvwxyz{|}~"
```

Figure 2-1-1-1 Template File Layout

2.1.1.2 Value File

Each Value file will contain the following information:



- (1) the (y,x) coordinate for the start of the value
/text,
- (2) the (y,x) coordinate for the end of the value
/text,
- (3) the value or text.

The (y,x) coordinate will be the character position on the IBM-AT screen with (0,0) representing the upper, left corner. X represents the row and Y represents the column position. Each coordinate value will be three characters in length.

The start coordinate will indicate where the text/value is to be placed on the display. The end coordinate will indicate where the text/value will finish, at most, on the display. These parameters will also be used when screen-editing, to be discussed in a succeeding section.

NOTE: The end coordinate will not be the position of the last character of the current value but, rather, the last coordinate of the largest text/value string.

Combined, the starting and ending coordinates will be used to determine the maximum number of characters that may make up a given field.

In some instances the value on a display may actually be a piece of text. For example, the text "GJ" references the TIMDS recording parameter. For this reason, text values must be supported in the Value file. All value fields will look like text fields when the value file is edited: the software will convert those fields as defined by the code to either hexadecimal or decimal integer values, or text strings, as required.



#2100-12-002.01.0

Figure 2-1-1-2 shows an example layout of a Value file.

003	019	003	025	"reverse"
004	019	004	021	"576"
007	025	007	019	"IBM-AT Memory"
008	014	008	018	"-2.45"

Figure 2-1-1-2 Value File Layout



2.1.1.3 File Names

The following conventions will be employed in the naming of the display data files:

- (1) Each file name will commence with 'dsp' standing for DiSPlay,
- (2) Template files will have the dot (.) extension 'tpl' (TemPLate) while value files will have the extension 'val' (VALue),
- (3) Each 'dsp' string will be followed by three numbers: these numbers will represent the option and suboptions chosen by the Operator. The file name for the Main Menu will be dsp000.tpl with no value file existing.

Examples:

- | | | |
|--|-----|------------|
| (1) Main Menu | | dsp000.tpl |
| (2) System Specifications (option 1) | | dsp100.tpl |
| (3) Option 1, suboption 2 | | dsp120.tpl |
| Recording Parameters | and | dsp120.val |
| (4) Option 1, suboption 2, suboption 2 | | dsp122.tpl |
| Page 2 | and | dsp122.val |

NOTE: if a display has no Operator-modifiable values in the Presentation Area, no value file will be needed and none will exist (for instance, dsp000.val is nonexistent).



2.1.2 Display Windows

The display windows, as discussed in the previous section, will be defined as types existing in the 'Screen Package' software of the VENIX Operating System. Figure 2.1.2 shows the C language definition of the four windows.

```
WINDOW *present_area;  
WINDOW *error_line;  
WINDOW *input_line;  
WINDOW *prompt_line;
```

Figure 2.1.2 Window Definitions



2.1.3 Presentation Area Field Values

When a display or menu is presented on the IBM-AT Monitor, all text for the various Presentation Area field values will reside in memory while the display is visible. The text will be stored in a character array with all field values run together as shown in Figure 2-1-3a. Each message will be defined in terms of a start and end pointer into this character string as illustrated.

A screen will consist of several fields. These fields will be linked to the left, right, up and down. Storage will be allocated for the pointers to the next fields.

Also needed will be the starting fields from the Input Line: that is, if cursoring starts at the Input Line where will a left arrow key move the cursor to? Storage will be allocated for each of the up, left, right, and down starting fields from the Input Line.

Storage will also be required to save the field of modification so that cursoring may be intermixed with other functions, as described in later sections.

All the fields will be collected in a table of a predetermined size. Within this table all next field pointers and screen location coordinates will be saved. The screen will consist of this table as well as the display text and the Input Line cursoring start pointers.

The C language data type and variable representations are defined in Figure 2-1-3b and Figure 2-1-3c, respectively.



#2100-12-002.01.0

NOTE: the end of the field table will be marked by an ending X
coordinate value of zero for the marking field.



display text																																								
0	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5															

G	G	1	0	0	0	0	1	2	1	7	G	I	-	2	2	3	0	0	G	J	5	1	0	0	2	...														

											^																													
											start															^														
																										end														

Figure 2-1-3a Display Text Representation

```

struct field
{
    int  lf;      /* field to left */
    int  rt;      /* field to right */
    int  up;      /* field above */
    int  dn;      /* field below */
    int  s_x;     /* field start X coordinate */
    int  s_y;     /* field start Y coordinate */
    int  e_x;     /* field end X coordinate */
    int  e_y;     /* field end Y coordinate */
    int  t_s;     /* index to start of display text */
    int  t_e;     /* index to end of display text */
};

```

Figure 2-1-3b Field Construct

```

struct scr_typ
{
    struct field  f_lay[MAX_FLD];  /* field layout */
    char  d_txt[DSP_TXT_MAX];      /* display text */
    int  i_lf;      /* 1st field left of Input Line */
    int  i_rt;      /* 1st field right of Input Line */
    int  i_up;      /* 1st field above Input Line */
    int  i_dn;      /* 1st field below Input Line */
    int  f_ptr;     /* field of modification */
} scr;

```

Figure 2-1-3c Variable Storage Definitions



2.2 Error Table

All text for the various error and status messages of the Field Checking System will reside in memory during execution of the System. The text will be stored in a character array with all messages run together as shown in Figure 2-2a. Each message will be defined in terms of a start and end pointer into this character string as illustrated in Figure 2-2b. This text will be read in form the IBM-AT hard disk.

Two other variables will be used to store the lowest and highest error and status codes available in the System (these values will be needed for error checking). Error messages will have negative indices into the Error Table and status messages will have positive positions. However, since C language arrays must begin at 0 and cannot be negatively indexed, error codes will have a fixed, constant offset added to them when referring to the Error Table via 'start' and 'end' pointers. For example, error code -4 with an offset of 40 will reserve Error Table position 36 for the start and end text pointers for message -4. The software routine which reads in the text from the disk and builds the Error Table will add this offset to the error number prior to storing the message.

The C language data type is defined in Figure 2-2c and the storage variable is defined in Figure 2-2d.

```
0 1 2 3 4 5 6 7 8 9 10 1 2 3 4 5 6 7 8 9 20 1 2 3 4 5 6 7 8
-----
/OE R R O R /O O P T I O N   N O T   A V A I L A B L E /O I
-----
      ^                                   ^
```

Figure 2-2a Error Text Storage



#2100-12-002.01.0

start end			ERROR TABLE

0	0	0	
1	1	6	
2	7	27	* 3 - 40 = Error Code - 37
3*	28	..	as read from disk file
..	

Figure 2-2b Error Table Pointers



#2100-12-002.01.0

```
struct msg_tbl_typ
{
    int  start[MSG_CNT]; /* start of string */
    int  end[MSG_CNT];   /* end of string   */
    int  min_code;       /* smallest code value */
    int  max_code;       /* largest code value  */
};
```

Figure 2-2c Message Table Type Definition

```
struct msg_tbl_typ  error_table;

char  text[ERR_TXT_LEN];
```

Figure 2-2d Variable Storage



2.3 Prompt Table

All text for the various prompts of the Field Checking System will reside in memory in the same form as error and status messages discussed in the preceding subsection. The C language variable definition is shown in Figure 2-3.

```
struct msg_tbl_typ  prmpt_table;  
char  pmt_txt[PMT_TXT_LEN];
```

Figure 2-3 Prompt Table Storage



2.4 Logical Unit Numbers

The Logical Unit Number (LUN) Table will map physical devices of the Field Checking System to a representative value or LUN. This table will resemble the example of Figure 2-4a. Note that LUN 0 is reserved for the NULL device and no reassignment is possible by the Operator. Specifying a LUN of 0, then, will signify that no such physical device is available. Figure 2-4b shows the C language data structure for the LUN Table and Figure 2-4c shows the storage variable.



#2100-12-002.01.0

no device*	0*	
mag tape 1	4	
mag tape 2	7	
hard disk 1	3	
hard disk 2	1	
Versatec printer	8	
Versatec plotter	6	*non user
IBM Monitor	2	definable
star printer	5	

logical unit number	physical device number	device descriptor	device name
0	0	0	/dev/null
1	4	0	/dev/hd1
2	8	0	"monitor"
3	3	0	/dev/hd0
4	1	0	/dev/mtr0
5	6	0	/dev/lp
6	7	0	/dev/spp2
7	2	0	/dev/mt1
8	5	0	/dev/lp2

Figure 2-4a LUN Table Representation



#2100-12-002.01.0

```
struct lun_tbl_typ
{
    char  phys_dev;      /* physical device no. */
    int   fp;            /* file descriptor */
    char  *dnam;         /* device name string */
};
```

Figure 2-4b LUN Table Type Definition

```
struct lun_tbl_typ  lun_tbl[LUN_TBL_LEN];
```

Figure 2-4c LUN Table Storage Variable



2.5 Recording Parameter Specifications

Each recording parameter specification will consist of seven parts: (1) the two-character parameter code used as an identifier when the data is recorded, (2) a parameter mnemonic or short-form version of the actual parameter name, to be used for printing and plotting, (3) the parameter type, referring to either a decimal or hexadecimal form, (4) the maximum number of characters/digits that will form the parameter value, to be used for parameter verification and data block formatting, (5) the parameter units expressed as text, (6) the parameter scaling value expressed as the number of decimal places in a raw data value, and (7) the sign associated with a parameter value, expressed as a single character.

Figure 2-5a illustrates the structure of one recording parameter specification. Figure 2-5b shows the C language type definition to be used to organize this information. Several of these specifications will be combined in a C language table as defined in Figure 2-5c.



char char	parameter code

	parameter mnemonic

char	type (decimal or hex)

char	length (# of digits)

	units (text)

char	scale (no. of decimal places)

char	sign

Figure 2-5a Recording Table Representation



#2100-12-002.01.0

```
struct  parms
{
    char  code[2];           /* parameter code */
    char  mnem[MNEM_LEN];    /* mnemonic */
    char  typ;               /* type - hex. or dec. */
    char  len;               /* length */
    char  unit[UNIT_LEN];    /* units (text) */
    char  scale;             /* no. decimal places */
    char  sign;              /* sign */
};
```

Figure 2-5b Element Type Definition

```
struct parms  rec_prm[REC_TBL_LEN];
```

Figure 2-5c Storage Variable



2.6 Airborne/Diurnal Data Character Sets and Headers

For plotting and printing diurnal and airborne data a character-checking set and header will be required. The character-checking set will simply be an array of characters, of a predefined length, specifying those characters which may be used within the particular data block.

The block header will also be an array of characters, similar to that found in Figure 2-6a, made up of spaces and parameter mnemonics. However, this header will be saved internally in a slightly different format. Each of the parameter mnemonics will be stored as references into the Recording Parameter Table. Along with each of these indices will be counts of the number of preceding spaces in the header. Figure 2-6b illustrates an example of the internal representation of the header. Figure 2-6c and Figure 2-6d show the C language data structure and variable representation, respectively, for each of the airborne and diurnal character-checking sets and headers.

```
                header
-----
TIMDS   .01GU   .01GL   MAGU1
-----
```

Figure 2-6a Sample of Airborne Print Header



	rec. parm table index	number of preceding spaces
0	4	2
1	20	3
2	21	3
3	8	5
4	0	0
..

Figure 2-6b Internal Representation of Print Header

```

struct  chk_set_typ
{
    char  char_set[CHAR_SET_LEN];
    struct
    {
        struct index_typ  p_idx;    (see Figure 2-8b)
        int  spaces;
    }  header[REC_TBL_LEN];
};

```

Figure 2-6c Check Set/Header Data Type

```

struct chk_set_typ  air_chk_set;
struct chk_set_typ  diu_chk_set;

```

Figure 2-6d Storage Variables



2.7 Storage Device Specifications

The Field Checking System storage devices will require data structures to maintain information during operation of the system. The two major storage media will be magnetic tape and flexible bernoulli disk drives.

Associated with each magnetic tape device will be three user specifiable attributes: (1) recording density, which is fixed for the drive, (2) interrecord gap which is also device dependent, and (3) length of usable tape which allows for differing length tapes to be used by the System.

The two Bernoullin disks will be treated as raw devices where data will be steamed onto them without elaborate file structures. As a result, the disks will be partioned by the Operator as a means of accessing only specific areas of the disk. These partions will be defined by logical disk addresses in conjunction with the disk directories contained on the disk.

The disks will have variable storage reserved for the disk directories which will contain the traverse line numbers, date, number of records for each block, and starting and ending logical disk addresses for each line on the disk. Figure 2-7a shows the C language data type for the storage device specifications, figure 2-7b shows the variable definition for this information, figure 2-7c shows an example of the information contained in a disk directory, figure 2-7d shows the data type for a disk directory, and figure 2-7e shows the variable definition for each of the two directories in the system.



```

struct sdev_typ
{
    char   dsk_intlv;          /* disk interleave */
    struct
    {
        unsigned   rec_dns;    /* recording density */
        unsigned   use_tap;    /* usable length of tape */
        float      int_gap;    /* interrecord gap */
    } tap[2];
};

```

Figure 2-7a Storage Device Specs. Type Structure

```

struct sdev_typ  sdv_spc;

```

Figure 2-7b Storage Device Specs. Variable

entry	line number	flight date	block size	start LAD	end LAD
0	00010127	03287	6	00006	00030
1	01262854	03287	6	00030	0014c
2	61963265	03287	6	0014c	00350
3

Figure 2-7c Disk Directory Representation

```

struct dir_typ
{
    struct
    {
        char   date[5];          /* flight date */
        char   bcnt;             /* no. recs/data blk */
        unsigned char   line[4]; /* line number */
        unsigned char   s_ld[3]; /* starting LAD */
        unsigned char   e_ld[3]; /* ending LAD */
    } entry[MP_ENTRYS];
};

```



#2100-12-002.01.0

```
struct dsk_map_typ
{
    long s_lad;      /* start LAD for disk partition */
    long e_lad;      /* end LAD for disk partition */
    long cur;        /* current LAD accessed on disk */
    char b_cnt;      /* no. sectors per data block */
    union
    {
        struct dir_typ dir;      /* internal rep. */
        char bfr[DSK_MAP_LEN];   /* raw disk map */
    } dd;
};
```

Figure 2-7d Data Types for Disk Directories

```
struct dsk_map_typ d_map0;
struct dsk_map_typ d_map1;
```

Figure 2-7e Disk Map Storage Variables



2.8 Plot Table

The Plot Table will contain the different parameters needed to create a plot of recorded data. Figure 2-8a illustrates the makeup of the Table.

The vertical scale will be saved as an integer, representing the number of blank lines to be left between horizontal plot lines.

The fiducial parameter (the plot reference) will be defined in terms of a pointer to the appropriate entry of the Recording Parameter Table. A storage location will also be reserved for the pointer to the parameter in the first record of a recording data block. These two pointers will be defined by the C language construct shown in Figure 2-8b. Each fiducial parameter will also require an interval value.

A character flag will save the response to whether or not printing of vertical scale lines is desired. Storage will also be made available for the source LUN.

For each parameter to be plotted, a pointer to the appropriate entry of the Recording Parameter Table will be needed along with a pointer into the data block. Included, also, will be the parameter plot interval along with the start and end sectors of the plot page within which the curve for each parameter must lie.

Finally, space will be reserved for a flag to indicate whether or not the plot is to be one of Successive Differences.

Figure 2-8c shows the C language definition of the Plot Table.



#2100-12-002.01.0

vertical						
scale						

fiducial		rec tbl		data blk		interval
parm:		index		index		

scale line						
flag						

source lun						

plot		rec tbl		data blk		scale
parms:		index		index		

successive						
diff. flag						

(36)

Figure 2-8a Plot Table Representation



```
struct
{
    int   r_tbl;    /* record. parm. table index */
    int   d_blk;    /* index to parm. in data block */
} index_typ;
```

Figure 2-8b Index/pointer Type Definition

```
struct plt_tbl_typ
{
    int   v_skl;                /* vertical scale */
    struct
    {
        struct index_typ ptr;  /* rec. parm ptr */
        long intvl;           /* interval */
    } fd_pm;                  /* fiducial parm */
    char skl_flg;              /* scale lines? */
    int   src_lun;             /* source device */
    struct
    {
        index_typ ptr;         /* plot parm */
        int skl;               /* scale */
        int s_sct;             /* start sector */
        int e_sct;             /* end sector */
    } pm[MAX_PLT_PRM];
    char s_dif;                /* succ diff flag */
} plt_tbl;
```

Figure 2-8c Plot Table Type Definition



2.9 Stacked Profile Table

Figure 2-9a shows the storage allocation required for the Stacked Profile Creation Process.

The first two fields of the Stacked Profile Table will contain the source LUN and the flight date that the data for the traverse lines was collected. This date will be used during the plotting process.

For each of the parameters which the Operator may specify to be extracted, the two pointers seen in preceding subsections will have space allocated for them (the Recording Parameter Table and data block indices). Two additional fields will be provided: (1) the parameter value, to be used during the actual plotting process and (2), a pointer into the first record for the parameter of the temporary data blocks formed from the extracted data. This temporary block will be explained in more detail in the section entitled "Stacked Profiles".

A single integer will be used to save the new, temporary record length formed from the extracted data.

The Flight Lines Table will contain the majority of the information needed to create a Stacked Profile. This table will be filled starting from index location 0, upwards, as new lines are encountered during preprocessing of the data.

The traverse line number will head the Table. This will be followed by two character fields used to define the orientation of the line data. When extracted from the source medium, the orientation will be assumed to be "back, forth, back, forth,



back, etc." in the order encountered for each line. However, after extraction the Operator will be given the ability to change the orientation for each line which may be different from the extraction orientation. These two different views will be saved.

A character field will be used to specify whether the line data is to be included in the profile.

Space will be left for the file pointer of the extracted line data file, open during profiling. This will be followed by the table index for the next line to appear on the plot. The number of blocks of data extracted for the line will be the last variable in the Line Table.

The Extraction Table index for the Profile Parameter will have a single, integer allocation for it.

Flags will follow in the Stacked Profile Table specifying whether a Free Format profile is requested, flight line text is to be included next to each curve, and scale lines marking plot sector boundaries are to be placed on the profile.

The index of the first line of the Line Table will require storage allocation, also. The plot scale, first sector starting location, first sector ending location, and plot separation will follow. The plot scale will refer to the profile parameter. The first plot position will define the start and end sectors within which the leftmost curve must lie. The separation will define the number of sectors separating the starting sector of line X and the starting sector of line X+1. It will be entirely possible for some curves to be defined with overlapping plot positions.



#2100-12-002.01.0

Finally, the vertical scale will be defined. The vertical scale will represent the number of blank lines to be left between horizontal plot lines.

A single, character field will mark the end of the table, providing an easy way for the end of the table to be determined when examined in its ASCII format. This marker character will be an asterisk ('*').

Figure 2-9b shows the C language definition of the Stacked Profile Table.



#2100-12-002.01.0

```
-----
| source lun |
-----
| flight date |
| of profile |
-----
| extraction | rec tbl | data blk | value | temp. data | (10)
| parameter: | index   | index    |       | blk index  |
-----
| new record |
| length     |
-----
| flight      | line    | extraction | user-defined |
| line #s:    | number  | orientation | orientation   |
-----
| print on |
| plot flag |
-----
| file      | next line | no. of blks |
| pointer   | on plot   | in line     | (24)
-----
| profile    |
| parameter  |
-----
| format     | rec tbl | interval |
| parameter: | index   |          |
-----
| free format |
| profile flag |
-----
| first line |
| index      |
-----
| scale lines |
| included flag |
-----
| list line nos. |
| text flag      |
-----
| plot          |
| scale         |
-----
| plot | plot | start | end | plot |
| specs: | scale | sector | sector | separation |
-----
| vertical |
| scale    |
-----
```



#2100-12-002.01.0

```
-----  
| end of table |  
| marker      |  
-----
```

Figure 2-9a Table Representation



```

struct
{
    int    src_ln;                /* source LUN */
    unsigned j_date;             /* flight date */
    struct
    {
        index_type ptr;          /* indices */
        long        val;          /* value */
        int         tb_inx;       /* new record index */
    } xtrct[MED_TBL_SIZ];        /* extraction parms */
    int tr_ln;                    /* new record length */
    struct
    {
        long ln;                 /* line number */
        char rvrs;               /* extracted orient. */
        char or;                 /* user defined orient */
        char pr_flg;             /* include in plot flag */
        int lp;                  /* file pointer */
        int nsq;                 /* next line on plot */
        long bk_cnt;             /* no. of blks in line */
    } ln_tbl[MX_F_LN];          /* line table info. */
    int pf_pm;                   /* profile parameter */
    struct
    {
        int pm;                  /* format indices */
        long ivl;                /* interval */
    } fmt;
    char free;                   /* free format flag */
    int f_ln;                    /* index to first line */
    char scl_ln;                 /* scale lines flag */
    char lst;                    /* list lines flag */
    struct
    {
        int p_scl;               /* plot scale */
        int p_ss;                /* first start sector */
        int p_es;                /* first end sector */
        int p_spn;               /* plot separation */
    } plt;
    int v_scl;                   /* vertical scale */
    char marker;                 /* end of tbl marker */
    } stk_tbl;

```

Figure 2-8b Type Definition



2.10 Gradient Parameter Creation Table

The Gradient Parameter Table will actually include three smaller tables. Figure 2-10a shows an example of these three tables. Up to nine gradient parameters may be created based on twelve defining parameters. For each defining and to-be-created gradient parameter, two values must be saved during compilation: (1) the pointer to the parameter in the Recording Parameter Table, and (2) the pointer to the data in the data block. A third item will be required for the defining parameters: the value of the parameter as obtained from the data block.

The "defining parameters" will merely be those specified by the Operator. The "pair parameters" will be those calculated from the difference between each pair of defining parameters. The "quad parameters" are the slightly more complex parameters found by taking the difference of averaged, compiled pairs. Included in the Gradient Table will be the source and destination LUNs of the two devices required by the process. As well, storage will be provided for a variable indicating the additional length of the new record containing the gradient parameters. Figure 2-10b shows the C language construct of the Gradient Table.



defining parameters				pairs		
0 (A)	3	23	..	0 (KG)	66	132
1 (B)	17	34	..	1 (KH)	67	140
2 (C)	14	67	..	1 (KI)	68	149
3 (D)	9	51	..	1 (KJ)	69	158
4 (E)	63	44	..	1 (KK)	-1	-1
5 (F)	9	60	..	1 (KL)	-1	-1
6 (G)	21	75	..	quads		
7 (H)	42	97	..	0 (KM)	72	166
8 (I)	-1	-1	..	1 (KN)	73	179
9 (J)	-1	-1	..	1 (KO)	-1	-1
10 (K)	-1	-1	..			
11 (L)	-1	-1	..			

			parameter value
			data block record index
			recording parm. table index

Figure 2-9a Gradient Table Representation

```

struct grd_tbl_typ
{
    struct def_typ
    {
        struct index_typ ptr;      /* data blk and rec.
                                   table ptrs */
        long val;                  /* data blk value */
    } def_prm[GRAD_DEF_CNT];      /* defining parms. */
    struct index_typ pr[GRAD_DEF_CNT / 2]; /* grad pairs prms */
    struct index_typ qd[GRAD_DEF_CNT / 4]; /* grad quads prms */
    int src_ln;                    /* source LUN */
    int dst_ln;                    /* destination LUN */
    int add_len;                   /* additional len */
} grd_tbl;

```

Figure 2-10b Gradient Table Type Definition



2.11 Edit Table

The Edit Table will contain the greatest collection of data of any of the tables. The Edit Table Representation is illustrated in Figure 2-11a. With the exception of the source and destination LUN's, all the data will be used solely by the Batch Editor.

For batch editing, start and stop limits are required. These values will be saved in the Table. A parameter upon which to base these limits is needed and will be specified by a pointer to the appropriate entry of the Recording Parameter Table. For quicker accessing during the edit, a pointer into the data block for each parameter will also be saved.

The source and destination devices will be specified as logical unit numbers (LUN's) and will be saved in the Table. These two entries will be used by both the Batch and Screen Editor.

To add a value to a parameter the two indices (Recording Parameter Table and data block indices) will be required as well as the value to be added. The same storage will be allocated for the specifications required to replace a parameter value. Only the indices will be needed to delete a parameter.

To replace a parameter value's digit, space will be reserved for the indices along with space for the digit position to be changed and the new digit, a character.

Spike checks will also have storage for the parameter indices as well as the spike definition. This definition will be the absolute change of the value over a given range.



To add a parameter to recorded data, several additional storage locations must be allocated to accommodate more than just the indices. For each new parameter, a calculation will be defined as a basis for the new parameter's creation. A table of operators and operands will make up this calculation. The operators will be the characters representing the computer symbols for the mathematical operations. The operand table will contain a tag indicating whether or not the operand is a constant or is the value of an existing recording parameter. If a constant, this value will be obtained from the numeric field of the operand table; otherwise, the numeric field will contain the index into the Recording Parameter Table for the specified operand and the data block pointer to the value.

Figure 2-11b shows the C language data type for one of the structures used to form the Edit Table definition of Figure 2-11c.



basis	rec tbl	data blk			
parameter:	index	index			
strt limit					
stop limit					
source lun					
destin lun					
add to	rec tbl	data blk	value	(10)	
value:	index	index			
replace	rec tbl	data blk	value	(10)	
value:	index	index			
delete	rec tbl	data blk	(5)		
parameter:	index	index			
replace	rec tbl	data blk	digit	digit	(10)
digit:	index	index	position	value	
spike	rec tbl	data blk	change	range	(10)
check:	index	index			
add	rec tbl	data blk			
parameter:	index	index			
	operands:	tag	(10)		
		value			
		OR			
		rec tbl	data blk		
		index	index		
	operators	(9)			

Figure 2-11a Edit Table Representation



```

struct
{
    struct index_typ ptr;
    long int value;
} val_typ;

```

Figure 2-11b Value Data Type

```

struct
{
    struct index_typ basis_parm; /* basis parm */
    int sr_lm; /* start limit */
    int sp_lm; /* stop limit */
    int sln; /* source LUN */
    int dln; /* destination LUN */
    struct val_typ a_vl[MED_TBL_SIZ]; /* add-to-value */
    struct val_typ r_vl[MED_TBL_SIZ]; /* replace value */
    struct index_typ d_pm[SML_TBL_SIZ]; /* delete parameter */
    struct
    {
        struct index_typ ptr; /* pointers */
        int d_ps; /* digit position */
        char d_vl; /* new digit */
    } r_dg[MED_TBL_SIZ]; /* replace digit */
    struct
    {
        struct index_typ ptr; /* pointers */
        int chg; /* absolute change */
        int rng; /* record range */
    } chk[MED_TBL_SIZ]; /* spike check */
    struct
    {
        struct index_typ ptr; /* new parameter */
        struct
        {
            char tag; /* type of operand */
            union
            {
                long val; /* constant */
                struct index_typ vptr; /* parm. variable */
            } vlu;
        }
    }
}

```



#2100-12-002.01.0

```
    } opnd[OPER_TBL_SIZ + 1];  
    char optr[OPER_TBL_SIZ];    /* operators */  
    } a_pm[SML_TBL_SIZ];  
} edt_tbl;
```

Figure 2-11c Edit Table Data Type



3.0 FULL SCREEN EDITING

When the prompt "USE CURSOR KEYS AND KEYBOARD TO SET UP SPECIFICATIONS" is issued on the Prompt line of the Display, the cursor keys on the keyboard will become enabled and the Operator will be permitted to do screen editing. Any changes made in the Presentation Area will be accepted as new values. Cursor movement will be as described in the next subsection.

3.1 Cursor Movement

The location or area of each modifiable value or text string shown in the Presentation Area of the screen will be referred to as a 'field'. Associated with each field will be a length, in characters, which is determined by the start and end field coordinates. When a display is presented and cursoring is enabled, the visible cursor will be noted at the Input Line. When cursoring is complete, the cursor should be returned to this position on the screen.

Horizontally, or left to right, cursor movement will be field by field. When the left and right arrows are depressed the cursor moves to the start of the next field to the left or right, respectively. If the cursor is at the beginning of a line and the left arrow key is struck, the cursor will be moved to the start of the rightmost field at the end of the line above. Similarly, if the cursor is at the end of a line, hitting the right arrow key will move the cursor to the start of the leftmost field at the beginning of the line below.

Vertically, cursor motion will also be by fields. When the up and down arrows are depressed the cursor will be moved to the



start of the nearest field directly above or below that one being vacated. If the cursor is at the top of a line and the up arrow key is struck, the cursor will be moved to the start of the next field to left of the vacated field, at the bottom of the screen. Similarly, if the cursor is at the bottom of a line, hitting the down arrow key will move the cursor to the start of the next field to the right of the vacated field, at the top of the screen.

When in the top, left field in the Presentation Area, depressing the left or up arrow keys will return the cursor to the Input Line of the screen. When in the bottom, rightmost field, the right arrow and down keys will return the cursor to the Input Line. Figures 3-1a through 3-1d illustrates the cursor movement over a sample Presentation Area for each of the cursor directions.

At any time the Operator may return to the Input Line by striking the 'enter' key.



```
+-----+
|               F I E L D   C H E C K I N G   S Y S T E M               |
|               |
|..> field1 ...> field2 ...> field3 ..> field4 .|
| . .....|
| .      V|
| . field5 ...> field6 .....> field7 .|
| . .....|
| .      .> field8 .|
| . .....> field9 .|
| . .....|
|-----|
| INPUT LINE <.....|
+-----+
```

Figure 3-1a -- Right Cursoring

```
+-----+
|               F I E L D   C H E C K I N G   S Y S T E M               |
|               |
|... field1 <... field2 <... field3 <.. field4 <.|
| . .....|
| .      .|
| . .field5 <... field6 <..... field7 <.|
| . .....|
| .      .field8 <.|
| . ..... field9 <.|
| . .....|
|-----|
| V|
| INPUT LINE .....|
+-----+
```

Figure 3-1b -- Left Cursoring



```

+-----+
|               F I E L D   C H E C K I N G   S Y S T E M               |
|                                                                           |
| ... field1 .... field2          field3 <.. field4 <. |
|      ^           ^              .              .      |
|      .           .              .              .      |
| field5<. .> field6          .          .> field7... |
|      .              .              .              .      |
|      .. field8 <.....          .              .      |
|                               .              .      |
|                               ..... field9          ^      |
|-----+-----+
v
INPUT LINE .....

```

Figure 3-1c -- Up Cursoring

```

+-----+
|               F I E L D   C H E C K I N G   S Y S T E M               |
|                                                                           |
| ..> field1 ...> field2          field3 ..> field4 .. |
|      .           .              ^              .      |
|      v           v              .              .      |
| field5.. .. field6          .          .. field7 <. |
|      .              .              .              .      |
|      .> field8 .....          .              .      |
|                               .              .      |
|                               .....> field9          .      |
|-----+-----+
INPUT LINE <.....

```

Figure 3-1d -- Down Cursoring



3.2 Field Modification

Within fields, movement will be character by character. Cursor keys, namely the arrow keys, will not be used for jumping between characters; only for fields. To replace a character in a field at which the visible cursor is located, the Operator need only enter the desired, new character. To move to the right within a field, without overwriting the displayed character, the 'tab forward' (-->|) key will be used. Continually striking the tab forward key will advance the cursor one position to the right until the end of the field is reached. At this point the bell will sound to indicate no further movement is possible when this key is hit.

To nondestructively backspace, the backward tab key (|<--) will be used. Continually striking this key will return the visible cursor to the start of the field at which time the bell will sound to signal no movement being possible.

A destructive backspace may be made using the delete (<--) key. Again, no backing up past the start of the field will be possible. The bell will sound if an attempt to delete past the start of the field is attempted.

To insert a blank character in the field, the open, square brace key ([) will be used. This key will simply make room for characters to be added at the cursor location. If two characters are to be inserted, the square brace key must be hit twice.

NOTE: The square brace key will not cause an "insert mode" to be entered but rather will make space available for inserting characters. One other important note is that the space made



#2100-12-002.01.0

available for insertion characters will cause all characters of the field to the right of the cursor to be shifted one position to the right. The last character of the field, then, will be lost every time the open, square brace key is depressed.

To delete a character in a field, the cursor will be positioned over the unwanted character and the closed, square brace key (]) struck. This will cause all characters to the right of the cursor to be shifted one position to the left. A blank (space) character will be added at the end of the field.

At any time, the Operator may return to the Input Line by striking the enter key (Enter <--').



4.0 SYSTEM SPECIFICATIONS

When the System Specifications option of the Main Menu is chosen, the System Specifications Menu will be displayed along with the prompt, "ENTER NUMBER OF OPTION". Keyboard entries will be handled until a CTRL Z or CTRL C is received to return the Main Menu display.

4.1 Logical Unit Numbers

Choosing to modify the logical unit numbers (LUN's) for the Field Checking System will result in the Logical Unit Numbers Display being presented. The prompt "USE CURSOR KEYS AND KEYBOARD TO SET UP SPECIFICATIONS" will be shown on the Prompt Line. Keyboard input will then be accepted. Input will first be examined for cursor motion keys and, if the key is one of the left, right, up or down arrows or the enter key, the screen will be updated accordingly. Otherwise, the value will be changed on the screen and stored for later processing.

When a CTRL Z exit is requested, verification of the values specified will take place. First, each LUN will be checked to ensure that it is a numeric value, secondly, that it is within the valid range of values and, thirdly, that no two logical unit numbers, with the exception of 0, are identical. Error messages of "INVALID CHARACTER", "INVALID DATA" and "DUPLICATE LUN's ASSIGNED" will be issued in either case, respectively.

If no errors are detected, the previous menu will be returned with a CTRL Z exit. Ctrl Z will save the modifications in the display's value file and will process the changes, updating the LUN table. CTRL C will ignore all changes on an exit.



4.2 Recording Parameters

Option 2 of the System Specifications will define the Recording Parameters comprising Airborne and Diurnal data. When the Recording Parameter Display and prompt "USE CURSOR KEYS AND KEYBOARD TO SET UP SPECIFICATIONS" are presented on the Monitor, the Operator will be free to make any changes desired to any of the three pages of parameters.

For each recording parameter number, there will be seven specifications as described in section 2.5. Because of the large number of recording parameters, three display pages will be required. To move back and forth between pages the Operator will type, while the visible cursor is located at the Input Line window of the display, the desired page number to be presented. Verification will be done immediately, before a valid, new page of parameters is presented. At this time, checks will be made for valid parameter codes, parameter names present, correct parameter types, valid parameter lengths, unique parameter codes, unique parameter names, valid parameter signs, and valid parameter scales. Corresponding error messages will be issued for any inconsistencies discovered. If no errors are detected, the new page will be displayed.

When the Operator exits from any page via CTRL Z, these same checks will be performed. If no errors are encountered, the Recording Parameter Table and value file will be updated with the new changes and a return to the System Specifications Menu will take place. If errors are found, however, the Operator will be forced to remain in the Recording Parameter Display until either all the errors are corrected and CTRL Z is entered or a CTRL C sequence is used to escape. CTRL C will cause all changes made



#2100-12-002.01.0

to all pages to be disregarded and the previous menu returned.



4.3 Airborne Data Character Set & Header

Choosing option 3 of the System Specifications Menu will result in the Airborne Data Character Set Display being presented on the IBM-AT Monitor. The Prompt Line will contain the text "USE CURSOR KEYS AND KEYBOARD TO SET UP SPECIFICATIONS".

Keyboard input will be accepted with cursor movement being examined first. If an arrow or edit key is encountered the display will be updated accordingly. Data keys will be accepted at face value: that is, no verification will take place until the Operator signals a return to the previous Menu via CTRL Z.

Upon receiving a CTRL Z, the Field Checking System will verify the character-checking set for uniqueness of characters. If duplicate characters are evident the message "DUPLICATE CHARACTERS FOUND" will be displayed.

Next, each word (characters separated by spaces) of the Block Print Header will be verified against the valid recording parameter mnemonic table entries. Inconsistencies will be flagged by the message "INVALID PARAMETER NAME" with the cursor positioned at the Header field. The Operator will then be expected to make the necessary correction before attempting to save again and return to the Main Menu. No updating will be permitted by the software until the character-checking set contains only unique characters and the block print header contains only valid mnemonics.

NOTE: the 'NULL' parameter must be placed at the end of the header to mark the last column position to be used during dumping; the error message "MISSING PARAMETER(S)" will be issued



#2100-12-002.01.0

if this is not done.

When both fields are free from error, a CTRL Z exit will cause the new values for both the Data Character Set and Airborne Block Print Header to be written to the corresponding value file for the display and the respective tables updated.

CTRL C at any time will return the System Specifications Menu with the modifications having been discarded. See the section entitled "Block Printing" for more on the character-checking set and header.



#2100-12-002.01.0

4.4 Diurnal Data Character Set & Header

The same procedure will be followed for the Diurnal Data Character Set & Header option of the System Specifications Menu as is performed for the Airborne Data Character Set & Header option.



4.5 Storage Device Specifications

When the fifth option of the System Specifications Menu is chosen the Storage Device Specifications Display will be presented along with the "USE CURSOR KEYS AND KEYBOARD TO SET UP SPECIFICATIONS" prompt.

Modification of these system specifications will be done in a similar manner as described in the previous subsections. Here, the Field Checking System will be looking to verify that the Disk Logical Addresses are valid hexadecimal values and that they encompass an increasing portion for each disk; and that the amount of usable tape, interrecord gaps, and recording densities for each drive are valid decimal values and within acceptable ranges. No exit and save will be done until any errors detected by the System have been corrected by the Operator.



5.0 MEDIUM CONTROL

Choosing the Medium Control option of the Main Menu will result in the Medium Control Menu and "USE CURSOR KEYS AND KEYBOARD TO SET UP SPECIFICATIONS" prompt being presented on the display. The Operator will choose to do either Tape Control, Disk Control, or Copy/Search Functions from the Menu.

5.1 Tape Control Functions

The Tape Control Functions Display will be presented when option 1 of the Medium Control Menu is chosen. Prior to choosing an option from this display, the LUN for the device to be accessed must be set in the Presentation Area.

Magnetic tape operations will be performed as specified. If an error of any kind is encountered, the appropriate message will be issued on the Error Line. Messages will include "INVALID CHARACTER", "INVALID DEVICE", "CANNOT OPEN DEVICE" and "ERROR AT SOURCE DEVICE". The auxiliary Operator's manual will describe the cause of these errors in detail.

A CTRL Z exit from this display will save the LUN specified as the new default, providing it is not in error. Leaving the display via CTRL C will discard any changes made in the Presentation Area.



5.2 Disk Control Functions

The second option of the Medium Control Menu will cause the Disk Control Functions Display to be presented. Prior to choosing an option from the display, the LUN for the device to be accessed must be defined, and, if a cartridge is to be formatted, the sector interleave must be set.

The first option will allow cartridges to be formatted. The second will allow the directory to be cleared on a cartridge. Choosing either of these options will result in a prompt being issued to ensure that the operation is truly desired. If the Operator responds affirmatively, the option will be attempted.

A third option will cause the disk directory to be read and presented on the display. This directory will show the data outlined in section 2.7.

Exiting the Menu will be similar to that for the Tape Control Functions.

5.3 Search/Copy Functions

The third option of the Medium Control Menu will allow storage media to be searched for strings and data to be copied between differing devices. Prior to performing an operation both a source and destination LUN must be specified in the Presentation Area of the display.

Searching for strings on a medium will require the Operator to set up the search string in the Presentation Area. When invoked, and after an affirmative response to the issued prompt "ARE YOU



SURE?", the search will continue until (1) either a match is made, (2) the end of the device is encountered or (3) an error occurs. In either of the three cases a hard-copy message will be issued on the standard error device (usually the Star matrix printer): in the first case the message will be to the effect that "a match was found on string S in block #B, at Cth character"; in the other two cases the message will resemble "no match found for string S". The date and time will also be issued. The standard error console will be used instead of the Monitor because of the need to retain a permanent record of this time-consuming operation. If the source device is one of the two hard disks, the console message will also include the logical address of the data block containing the match.

Similarly, magnetic tape may be positioned to a desired location using a special option of this Menu. The Operator's Manual will describe this operation in detail.

Data copy between media will be performed until successfully completed or an error is encountered. To prevent accidental copies, a prompt to the effect "ARE YOU SURE?" will be issued by the System prior to attempting to perform the copy option. Note the importance of specifying LUN's for this operation.

Exit from this Menu will be as described for the previous Control Functions.



6.0 BLOCK PRINTING

The Block Printing Process will allow the Operator to obtain a hardcopy of acquired data. Printing may be done to either the Star Matrix Printer or the Versatec Printer, with up to 132 characters of text per line.

When the Block Printing option of the Main Menu has been selected, the Block Printing Display will be brought up on the IBM-AT monitor. From the display the Operator will choose the print format. After the choice is entered by way of the keyboard, the Print Frequency Menu will be brought up.

The prompt "USE CURSOR KEYS TO SET UP SPECIFICATIONS" will be issued on the Prompt Line. The required specifications will be the source LUN of the data and the destination LUN for printing as defined via the System Specifications option; and a response indicating whether or not error checking is to be performed during dumping. When the Operator is satisfied with the specified values on the screen the the cursor will be returned to the Input Line.

The Operator will then enter the desired option listed in the Menu. If every N'th block is to be printed, the Operator will be prompted for the value of N. This value will be verified and if not valid, an error message will be issued. The Operator will then be expected to rechoose an option and, if the Every-N'th-Block choice is made again, the prompt for a value will be reissued. Until a valid option and, if necessary, a valid frequency, is specified, the Operator will remain in the Print Frequency Menu and no printing will be done.



#2100-12-002.01.0

After a valid option (and parameter) have been specified, Printing will commence.

The Header, set up in the System Specifications, will be used to set up the spacing for the block data as it appears in columns.

Recording blocks that are made up of many parameters may require more than one 132-character line to print a record in hardcopy form. When a record extends beyond a single line, the remaining parameters will be printed on a second line as defined by the block header.

Below the Header on the hardcopy will be a single, dashed line with a blank line below this. Above the Header will also be a blank line. Above this blank line will be a block title consisting of the block number and the block length. For Diurnal and Airborne data, the Julian date of recording will also be included in the Block Title. The traverse line number will be added for Airborne data.

Unformatted data will refer to a form in which the data is printed as a physical block of the actual length. Printing in this form will mean that no spaces will be added, no structuring will be done and no header will be provided. Unformatted printing will be required for nonaeromagnetically or nondiurnally acquired data. This form will also be used to print blocks containing bad data.

Bad data will be the result of characters appearing in the data block that do not appear in the character-checking set, blocks of differing lengths, records of differing lengths, or parameter values differing in type or length specified.



#2100-12-002.01.0

Printing will continue until terminated by the Operator, a file mark is detected -- in the case of magnetic tape -- or the end of a data map is encountered -- in the case of hard disk --, or an error occurs at either of the source or destination devices. When Printing is complete, CTRL Z and CTRL C will take the Operator out of Block Printing and back to the Main Menu with similar characteristics for saving/discarding changes as described for previous operations.



7.0 PLOTTING

When the plot option of the Main Menu is chosen, the Plot Specifications Menu and prompt "USE CURSOR KEYS AND KEYBOARD TO SET UP SPECIFICATIONS" will be presented on the display.

7.1 Plot Specifications

7.1.1 Standard Plot

Up to thirty-six recording parameters to be plotted may be handled by the system. For each parameter, specified by its two-character code, a plot scale and plot position will be required. The parameter code set up on the display will be verified against those found in the Recording Parameter Table. If no match is found, the error message "INVALID PARAMETER CODE" will be issued on the Error Line; otherwise the Table index will be saved.

The plot scale will be a long integer not to exceed eight digits in length: for hexadecimal parameters, the scale must still be expressed as a decimal value. The scale will be converted from the text seen on the screen to a decimal integer by the software. If any invalid characters are found during the conversion, an appropriate error message will be issued; otherwise the value will be saved.

The plot position will consist of two, three digit values representing the starting and ending sectors, inclusively, between which the plot is to be placed on the output device. A plot sector is defined as sixteen bits wide with a plot line of output containing 132 sectors numbered 0 through 131. Therefore,



a start sector of 3 and an end sector of 5 will provide a resolution of $3 \times 16 = 48$ bits for plotting the specified parameter. The bit to be turned on in a plot line will be determined by the following equation:

$$\text{plot bit} = \text{start sector} \times 16 + \\ \left(\text{value MOD scale} \times \right. \\ \left. \left(\text{end sector} - \text{start sector} + 1 \right) \times 16 \right) \\ \text{DIV scale}$$

examples:

scale = 45, start sector = 3, end sector = 6

$$\begin{aligned} (1) \quad \text{value} = 0 \quad & 3 \times 16 + (0 \text{ MOD } 45) \times (6 - 3 + 1) \times 16 \text{ DIV } 45 \\ & = 48 + (0 \times 64) \text{ DIV } 45 \\ & = 48 + 0 \end{aligned}$$

$$\begin{aligned} (2) \quad \text{value} = 42 \quad & 3 \times 16 + ((42 \text{ MOD } 45) \times 64) \text{ DIV } 45 \\ & = 48 + 59 \\ & = 107 \end{aligned}$$

$$\begin{aligned} (3) \quad \text{value} = 44 \quad & 3 \times 16 + ((44 \text{ MOD } 45) \times 64) \text{ DIV } 45 \\ & = 48 + 62 \\ & = 110 \end{aligned}$$

Also required, by Operator set up, will be a response to five other specifications. First, a vertical scale must be defined. This will be restricted to a two-digit value between 0 and 10. Specifying a value larger than ten will result in an error message, "LIMIT ERROR". The message "INVALID CHARACTER" will be displayed if nondecimal digits are found during the conversion from text to integer. A valid scale will be saved in the Plot Table.

Secondly and thirdly, a fiducial parameter and interval must be defined. The fiducial parameter, used as the scale along the leftmost edge of the plot, will be verified for existence by a search of the Recording Parameter Table. If a match on a



parameter code is not found, the message "INVALID PARAMETER CODE" will be displayed on the Error Line of the monitor; otherwise, the index will be saved in the Plot Table.

The interval will be converted from text to either a decimal or hexadecimal integer value (depending upon the parameter type) and any invalid character found will result in the presentation of a corresponding error message.

The Operator must specify whether vertical scale lines are desired. These lines will run vertically along the plot marking each start/end sector boundaries. Only a 'Y' or 'y' for the affirmative response or, 'N' or 'n' for the negative response will be accepted by the System: anything else will cause the message "INVALID RESPONSE" to be displayed.

Finally, the source lun must be specified. This must be one of the magnetic tape or hard disk devices to be acceptable by the System. If not, the message "INVALID SOURCE LUN" will be presented on the Error Line of the display.

7.1.2 Successive Difference Plot

Successive Differences, a special form of plot, will also be set up from the Plot Specifications Display. The parameter to be plotted will be specified as the first parameter in the upper, left corner of the Presentation Area. To distinguish between a regular plot and a successive difference plot, the scale and plot positions for the latter will all be defined on the screen as zero. If the System finds that the scale is zero but both plot position sectors are not, then it will be assumed that a regular plot is desired and an error message "INVALID DATA" will be



issued, referring to the missing scale value.

If the parameter for a Successive Difference plot has been specified correctly, then parameters two through eight will be examined as these define which Successive Differences are required. Only those seven parameter codes defined in the Recording Parameters Specifications relating to Successive Differences may be specified in any of the seven screen locations mentioned. At least one must be set up. A Successive Difference of zero will plot the actual data.

As with a regular plot, a scale and plot position must also be defined for Successive Differences with verification identical to that of regular plot specifications.

7.2 Plotting Process

If any errors are found during plot specification verification, the appropriate error message will be displayed and the field causing the error will be highlighted by positioning the visible cursor at the field in question. No plotting will commence until all errors have been corrected.

Once plotting is requested (via 'P' at the keyboard), it will continue until either the end of the source device is reached or a serious error occurs. A Plot Header will be printed at the top of the output with every new traverse line encountered causing a form feed to be issued and a new header generated for the next line.

Every time the current record value for the fiducial parameter is greater than the last saved fiducial value plus the fiducial



interval, a scale mark will be placed on each edge of the plot. Every tenth time, the actual fiducial value will be placed at the left edge of the output.

Regular plots will be generated by simply using the values from each record for each parameter to plot a line of data. Successive Differences, however, will require that data from several records be collected to form one plot line. For a fourth successive difference, for instance, a value will not be available for plotting until after the fifth record is examined. The software will maintain a table of the required number of previous data values for the parameter. When the correct number of values are available, the succeeding record value will be added and the oldest value removed from the table to create the new Successive Difference value.



8.0 STACKED PROFILES

Choosing option 5 of the Main Menu will bring up the Stacked Profile Display and the prompt "USE CURSOR KEYS AND KEYBOARD TO SET UP SPECIFICATIONS" on the Monitor. Several procedures will be performed prior to creating a profile.

8.1 Data Extraction

Necessary raw data from a source device must be extracted and organized as the first process of Stacked Profile creation. The System will allow up to ten parameters for extraction, one of which should be TIMDS.

Each parameter will be specified by the appropriate, two-character parameter code. Verification will be done on those codes specified to ensure their existence and definition in the Recording Parameter Table.

The only other specification required for the extraction process will be the source LUN. An invalid LUN will be flagged by an error message. No extraction will take place until all parameter codes are valid and the source LUN is one of the magnetic tape or hard disk devices.

Extraction will be initiated by the Operator depressing the capital 'P' on the keyboard. When extraction begins, a status message to this effect will be presented. Extraction will proceed as follows:

- (1) The header of each block will be extracted.
- (2) The requested data for each record of each



block will be extracted.

- (3) The header and extracted data will be combined to form a new, smaller block.
- (4) This new block will be written sequentially to a file entitled by the the flight line number of the data.
- (5) When data for a new flight line number is encountered, the current flight line file will be closed and a new flight line file will be opened.
- (6) The first, third, fifth, etc. lines extracted will be marked with an orientation of '1'; the second, fourth, sixth, etc. lines extracted will be marked with an orientation of '0'.
- (7) Steps (1) through (6) will be repeated until either an error is encountered or the end of the source device is reached.

When extraction is complete, a status message to this effect will be displayed. The Inclusion Process will then be performed.

8.2 Inclusion Process

The System will next request confirmation of the traverse line data to be included in the Profile based on the data extracted. The line number of each curve to be plotted will be specified in the Presentation Area. Along with each line number, the Operator will be provided a field in which to inform the System as to the true orientation of the line. If no orientation is provided on the display, the System will use that determined at time of extraction. However, any Operator-specified orientation will override the extraction value.



The only other item to note is that the System will plot the curves in the order that the lines are specified on the Inclusion Display.

When all inclusion specifications have been set up as desired, the Operator will move to the next process by typing a capital 'P' at the keyboard. If line numbers are specified for which no data was extracted then an appropriate error message will be issued on the display to inform the Operator of this fact and the error will have to be corrected before the Profiling procedure can continue.

8.3 Plot Profiling Process

Several plot specifications must be defined before the Profile can be generated. The first will be the plot/profile parameter. Again, this will be the two-character code, verified for error against the Recording Parameter Table. As an added check, the parameter will also be verified as having been extracted. If it has not, the message "MISSING PARAMETER" will be displayed on the Error Line.

Next, the format must be defined. If a free format or time-adjusted format profile is desired, the System will look to see that the TIMDS parameter has been extracted. If the profile is to be based on another format, the format parameter must have been specified for extraction. In either case, if the required parameter has not been extracted then an appropriate error message will be issued.

The Operator will inform the System whether or not to include line number text next to each curve. This will be done by a



character field in the Presentation Area.

Finally, the remaining four specifications, with the exception of "Separation" will be identical to, and handled the same as, those in the Plotting Process. "Plot Scale", "First Plot Position" and "Vertical Scale" will be defined. "Separation" will be the number of sectors separating each flight line curve on the profile output. A value between 0 and 131 will be accepted: anything else will result in an error message.

If errors are detected in the specifications, the Operator will be forced to correct them prior to the Stacked Profile creation. Once started, the profiling will continue until the end of all the temporary traverse line data files are reached or an error is encountered.

When profiling commences and completes, messages to these effects will be presented on the Error Line of the Display.

8.3.1 Free Format Profile

To create the Free Format Profile the parameter data will simply be extracted from each of the flight line data files a record at a time. Data will be extracted sequentially from lines with an orientation of '0'; data will be taken from the ends of files for those lines with an orientation of '1'. As each file empties it will be closed and the curve will no longer be plotted for the flight line. When all files are empty, the plotting terminates.



8.3.2 Time-Adjusted Profile

To create the Time-Adjusted Profile, the longest flight line will be determined. The absolute value of the recording time of the first record of the file subtracted from the recording time of the last record of the file will define the time period over which all curves will be scaled against. Shorter length files will be plotted according to the following equation:

$$\begin{aligned} \text{Time to plot yet} = & \# \text{ of records in test file } \times \\ & \text{plot line number DIV} \\ & \# \text{ of records in longest file } - \\ & \# \text{ of records in test file } \times \\ & (\text{plot line number} - 1) \text{ DIV} \\ & \# \text{ of records in longest file} \end{aligned}$$

This will simply be a vertical scaling similar to the horizontal scaling done to plot a parameter within its start and end sectors.

8.3.3 Other Format Profile

"Other" Format Profiles will be generated according to the following algorithm:

- (1) Read the first record of each flight line file
- (2) Use the smallest value as the start of the "other" scale parameter
- (3) Plot the smallest value and any that are equal to it
- (4) Get the next record for any flight lines just plotted
- (5) If the flight line file is empty close the



file

- (6) Add the plot interval to the value lasted plotted and call this new value the "smallest value".
- (7) Repeat steps (2) through (6) until all the flight line files have been closed

At varying intervals the value of the "other" parameter will be printed along the edge of the plot for reference. If line-number-listing is desired, these will be found at the top of the profile.

8.4 Exiting and Saving Stacked Profile Data

Similar to other operations, CTRL Z and CTRL C may be used to backup to the Main Menu. However, if extraction was successfully completed during the Profiling Process, before returning to the Main Menu a prompt will be issued asking whether to remove the extracted data from the IBM-AT Hard Disk. If the Operator does not remove the data, then it will be available the next time the Stacked Profile option is chosen. The internal Stack Table will also be saved on the IBM-AT Disk.

NOTE: To reuse data already extracted, a LUN of 0 will be placed on the Presentation Area field requesting the source device prior to initiating the Extraction Process.



9.0 GRADIENT PARAMETER CREATION

Choosing the Gradient Parameter option of the Main Menu will cause the appropriate display and prompt to be presented on the Monitor. The Operator will specify the recording parameters to be used in creating the new parameters, The two-character parameter codes will be expected: errors will be flagged with the message "INVALID PARAMETER CODE" if the parameters specified cannot be located in the Recording Parameter Table.

To create gradient parameters there must also be 2 to the power of X recording parameters specified. If not, the system will respond with the message "MISSING PARAMETER(S)".

Also required are the logical unit numbers for the source and destination devices. Both must be unique and be one of the magnetic tape or hard disk devices: if not, either of the messages "INVALID SOURCE LUN" or "INVALID DESTINATION LUN" will be presented on the Error Line of the display.

Gradient Creation will not commence until all specifications are error free. When this is so, an appropriate message will be issued on the display.

Each data block on the source device will be read and the requested gradient parameter codes will be appended to each record of a new, larger data block. Necessary, existing parameter values will be converted from text and used to calculate the new parameter's value which will then be converted to text and placed in the new data block. The new data block will then be written to the destination device with the process continuing until either the end of the source device is reached



#2100-12-002.01.0

or an error is encountered at either of the source or destination devices.

A check will also be done each time an existing data block is read to ensure that all required values are present and correctly located in the block.

A successful completion of the process (end of the source device is reached) will be marked by the presentation of an appropriate message.



10.0 EDIT FUNCTIONS

Choosing the Edit Function option of the Main Menu will result in the Edit Function Menu and the prompt "USE CURSOR KEYS AND KEYBOARD TO SET UP SPECIFICATIONS" being presented on the Monitor. When a valid choice is made, the appropriate subroutine will be called to handle the function desired. Prior to choosing one of the Menu options, the source and destination LUN's must be set.

Before the Edit Function Menu is brought up, the Edit Table will be initialized to an empty state. Once valid Table entries have been defined, these values will remain until an exit to the Main Menu is made.

10.1 Batch Editor

10.1.1 Start/Stop Edit Limits

This option must be called before a batch-edit can take place. A basis parameter must be specified first. This will be the two-character code of the parameter defining the start/stop limits. The Recording Parameter Table will be parsed to verify the existence of this parameter. If it does exist, the index into the Table will be saved in the Edit Table; otherwise, the message "INVALID PARAMETER CODE" will be issued.

Provided a valid parameter code has been specified, the start and stop edit limits will then be checked according to the parameter type, be it hexadecimal or decimal. If both are correct, they will be saved in the Edit Table. If not, an appropriate error message will result.



If a CTRL Z exit is made and no errors are found, these new specifications will be added to the Edit Table for later use.

10.1.2 Output NNN Blocks

This option will provide the Operator with an easy means of directly copying data that is to be left unchanged, between two devices. When the choice is made, a prompt "ENTER NUMBER OF BLOCKS" will be placed on the Prompt Line. The Operator will enter a valid, signed decimal number which will be verified. If okay, this number of blocks will be copied from the source device to the destination device provided no errors occur at either of the two devices. A message will be printed on the standard error console (usually the Star matrix printer) indicating how many blocks were dumped. This will provide a hard copy record for the Operator.

10.1.3 Add/Subtract a Value to a Parameter

To add a value to a parameter, the Operator will set up the two-character code to be used to identify the parameter in the data block. A comparison will be made against this code and the Recording Parameter Table to ensure the parameter exists. If it does not, the message "INVALID PARAMETER CODE" will be issued. Otherwise, the index into the Table for the parameter code will be stored in the Edit Table.

For each parameter, the value to be added must be specified. The value must match the type of the parameter, be it decimal or hexadecimal. If not, an appropriate error message will be given.



#2100-12-002.01.0

To subtract a value from a decimal parameter, the Operator will simply place a negative sign in front of the value. That is, the negative value will be added. For subtraction of a hexadecimal value, the ones-complement will be provided as the specification. The value to be added will be placed in the Edit Table.

Up to ten parameters may be specified for this option.

10.1.4 Replace a Parameter Value

To replace a parameter's value, the Operator will set up the two-character code to be used to identify the parameter in the data block. A comparison will be made against this code and the Recording Parameter Table to ensure the parameter exists. If it does not, the message "INVALID PARAMETER CODE" will be issued. Otherwise, the index into the Table for the parameter code will be stored in the Edit Table.

For each parameter, the replacement value must be specified. The value must match the type of the parameter, be it decimal or hexadecimal. If not, an appropriate error message will be given. The value to be added will be placed in the Edit Table.

Up to ten parameters may be specified for this option.



10.1.5 Delete/Insert A Parameter

10.1.5.1 Deletion

To delete a parameter from a recording block, the Operator will simply place the appropriate recording identifier next to one of the five deletion fields of the Delete/Insert Parameters Display. The recording identifier will be verified against those in the Recording Parameter Table to ensure its existence. The index into the Recording Parameter Table for the entry will be saved in the Edit Table for later reference.

10.1.5.2 Insertion

To insert a parameter, the Operator will enter the two-character code to be used to identify the parameter in the data block. A comparison will be made against this code and the Recording Parameter Table to ensure the parameter already exists.

IMPORTANT: before using the editor to add new parameters to data blocks, the Recording Parameter Specifications must first be updated to include the new parameter codes, as this is the only place in the Field Checking System that parameter codes may be added. Although the calculation defining the parameter is made in the Batch Edit option, the parameter reference must be made in the System Specifications option.

For each new parameter, a calculation must be defined in order for a value to be generated during batch-editing. This calculation may contain one of four operators: '+' for addition, '-' for subtraction, '*' for multiplication, and '/' for division. Only two types of operands will be permitted: (1)



recording parameter codes and (2) signed, decimal constants. The values for the given parameter codes will be substituted from each record during the edit process. Signed constants are specific values. Up to ten (10) recording parameters or numeric symbols may be included in one calculation.

As the operands are parsed and verified, they are stored in the Edit Table. A tag field in the table will identify the integer stored as either a constant or as an index into the Recording Parameter Table. This will facilitate faster computation during the editing process. The Operators will be stored as characters.

Up to ten parameters may be created at a time using this option.

10.1.6 Change One Digit

The appropriate display will be presented upon which the Operator will define up to ten (10) parameters to be changed along with a two-digit number indicating which digit to change and a one-character value of the new digit. When the parameters have been defined on the screen, each recording identifier will be verified against those contained in the Recording Parameter Table for existence. The index into the Table for the parameter code will be stored in the Edit Table for later reference.

Each 'digit position' specified will be examined for only decimal values. Each 'digit value' will be examined for the type of digits expected based on the specified parameter type found in the Recording Parameter Table. If errors are found, an appropriate error message will be issued; otherwise, these two specifications will also be saved in the Edit Table.



10.1.7 Check For Spike

To check for a spike the Operator must specify the parameter to be examined, the absolute change in value which will cause alarm and the range over which the change will apply.

A two-character parameter code will define the parameter. This code will be matched against those of the Recording Parameter Table and, if found, then the index will be stored in the Edit Table; otherwise, the error message "INVALID PARAMETER CODE" will be displayed on the Error Line of the Monitor.

The absolute change specification will be up to an eight digit value representing a fraction of the parameter values to be examined. If an error is determined it will be indicated, else the value will be saved in the Edit Table.

The range will define, in number of records, the difference for value comparison. For instance, a range of three would mean the 1st and 4th record values would be compared, the 2nd and 5th etc.. This range must be two decimal digits between 01 and 99; if not, an error message "LIMIT ERROR" will be issued. A correct value for the range will be stored in the Edit Table.



10.1.8 Perform Edit Changes

This option will process all the batch-editing specifications that have been set up using the preceding options. When batching commences a status message to this effect will be sent to the display. The source and destination devices will be opened and, if no errors have been encountered, the process will continue.

Each block of data on the source device will be read and checked against the edit limits. If not within the limits, the block will simply be copied to the destination device. If the basis parameter value is within the limits then editing will be carried out as follows for each of the ten records of the block:

- (1) The spike-checks will be done first. For each parameter to be checked the value of the parameter will be converted from text to an integer value. If "range" previous records have been examined then the spike test will be done according to the formula:

$$\text{change} = \text{ABS} (\text{current spike table entry} - \text{value at range previous entries})$$

If the change is greater than or equal to the limit change specified then a message will be logged on the standard error console to the effect that a "spike at block B, record R, value V, absolute change P" occurred. This will provide the Operator with a hard copy listing of all spikes detected within the limits. The current record value will be added to a circular queue, displacing a previous value if "range" records have been checked.

- (2) The record of the data block will be transferred to a temporary record buffer.
- (3) The change-one-digit edits will be performed next. This is a straight-forward text overwrite of the given digit in the temporary record buffer.



- (4) The replace-a-value changes will be made. The specified value will be converted to text and placed in the temporary record buffer.
- (5) The add/subtract-from-a-value changes will be made. The text value will be converted to its associated type and the specified value will be added to it. The new value will be converted back to text in the temporary record buffer.
- (6) The add-parameter changes will be performed next. The calculation specified will be performed as given. Checks will be done for missing parameters and divide by zero attempts. If the calculation is error free, the new parameter code and value, as ASCII text, will be appended to the end of the temporary record buffer.
- (7) The delete-parameter changes will be made. Those parameters to be deleted will have a special character placed in the parameter code and value fields of the temporary record buffer.
- (8) The temporary record will be copied to a temporary block buffer, less the special delete characters.

When the entire block has been edited (ie. all ten records) the new data block will be written to the destination device.

The batch editor will continue, successfully, until the stop edit-limit or the end of source device is reached or, unsuccessfully, until the end of the destination device is encountered or an error is detected. Completion of the editor will be specified by a message on the Monitor.



10.2 Screen Editor

The Screen Editor will allow blocks of data to be modified more intensively than with the Batch Editor, though the blocks must be modified individually. When the Editor is first invoked, a blank Presentation Area will be evident. Until a block of data is read from the source device no data will be displayed.

10.2.1 Reading a Block

To read a data block from the source device the Operator will depress the <CTRL R> keys. With this action a block of data will be read and the first 78 x 20 = 1560 characters will be written to the Presentation Area of the display. If 1560 characters are not available then only those characters that are [available] will form the remainder of the display.

10.2.2 Paging

10.2.2.1 Up

<PgUp> will cause the next string of 1560 characters to be presented. If this number of characters is not available then only those that are will be displayed: spaces will form the remainder of the display. If the Editor is in the insert-mode, however, no paging will be done. The bell will sound to indicate an erroneous action and the insert-mode will be exited.



10.2.2.2 Down

<PgDn> will cause the previous string of 1560 characters to be presented. Again, if this number is not available then only those that are will be displayed. If the Editor is in the insert-mode the bell will sound, no paging will be done and the insert-mode will be exited.

10.2.3 Cursoring

Prior to cursoring or doing any operations in the Presentation Area, the <Home> key may be struck to move the visible cursor to the top, left corner of the Display. Any of the four cursor keys will then allow movement over the displayed text. If a boundary is reached and an attempt is made to go beyond it, the bell will sound to signal the error.

If the cursor lies within the Input Line then cursoring will be as follows: depressing the <right> or <down> arrow keys will cause the cursor to be moved to the top, left character of the Presentation Area; depressing the <left> or <up> arrow keys will cause the cursor to be positioned at the lowermost, right character of the Presentation Area.



10.2.4 Destructive Changes

Any noncursor or noncommand key hit while outside of the insert-mode will replace the character at the visible cursor and the cursor will be moved one space to the right, if possible; otherwise to the beginning of the next line below, if possible; otherwise no movement will be made (the cursor will be located at the bottom, right corner of the Presentation Area in this case). Destructive screen changes will be made to the corresponding character of the data block, also.

10.2.5 Insertion

<Ins> will place the Editor in and out of the insert-mode. When inserting, the character entered at the keyboard will be placed at the screen location of the visible cursor, the cursor will be moved to the right, if possible, and all characters to the right of the new character will be moved right and/or down one position, if possible. Characters moved off the screen are not lost: a Page Up will return the character to the top of the screen.

To delete a character while in the insert-mode the backspace key is used. Backspacing can be done all the way to the start of the data block on the screen. This operation is destructive: characters backed over will be replaced by spaces both on the screen and in the data block.

If an insert beyond the bottom, right corner of the Presentation Area is done a Page Up operation will be done by the software and the cursor will be repositioned near the top of the screen as will the data being inserted. Similarly, backspacing past the



top, left corner of the Presentation Area will cause a Page Down sequence and the cursor will then be found near the bottom of the Presentation Area.

10.2.6 Deletion

Outright deletion of a character with a corresponding shortening of the data visible will be done using the key. When the cursor is positioned over the character to be removed and the key is depressed, the character will disappear and all characters to the right of it will be moved to the left by one position.

10.2.7 Backward One Block

<CTRL B> will back up the source device by one block, up to the beginning of the medium. This will be true for the hard disks, as well, where a 'seek' will be performed to do the backward block operation.

10.2.8 Error Checking

<CTRL E> will perform an error check of the block of data displayed on the IBM-AT Monitor. The error checking process will use the Airborne Block Header and Character-Check Set of the System Specifications to verify block length, character composition of the block, parameter codes and the digits of each parameter. Errors detected will be highlighted on the display using inverse video and the display will be redrawn so that the error is found in the top, left corner of the Presentation Area. Only one error at a time will be indicated: therefore, it may be necessary to use <CTRL E> several times to find and correct



#2100-12-002.01.0

several errors.

10.2.9 Writing a Data Block

<CTRL W> will write the edited version of the data block to the destination device. CTRL R will be required to get a new block.

10.2.10 Exiting Editor

<CTRL Z> or <CTRL C> will return the Operator to the Edit Menu with any changes made for the data block displayed on the screen left unrecorded.



SECTION 2

DEPENDENCY DIAGRAMS

The diagrams of the following section depict the procedural relationships of those modules written specifically for the Field Checking System. The numbers in brackets under the procedure names refer to the figure in which the dependencies for the routine in question are defined.

NOTE: many of the routines make calls to the Operating System library functions: however, these references have not been listed to avoid adding unnecessary confusion.



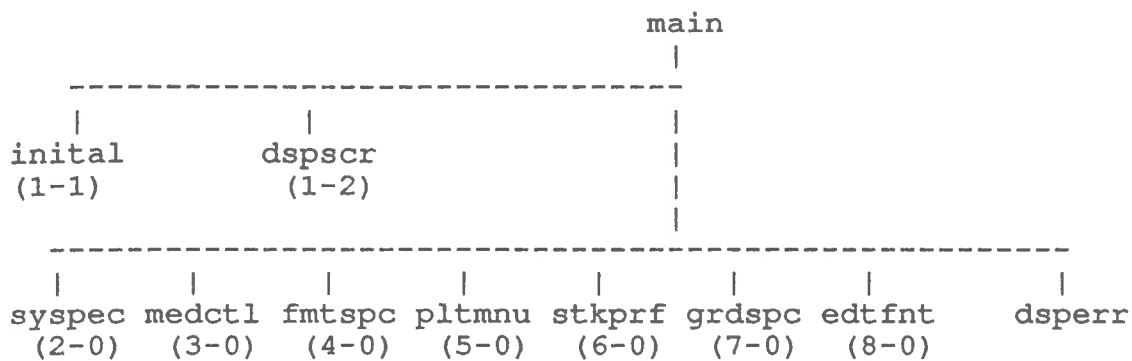


Figure 1-0

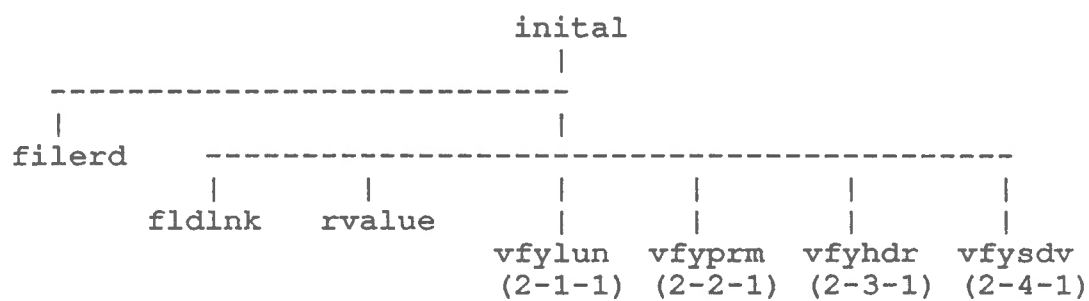


Figure 1-1

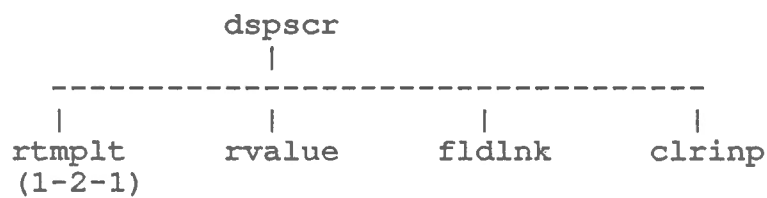


Figure 1-2





Figure 2-1-1

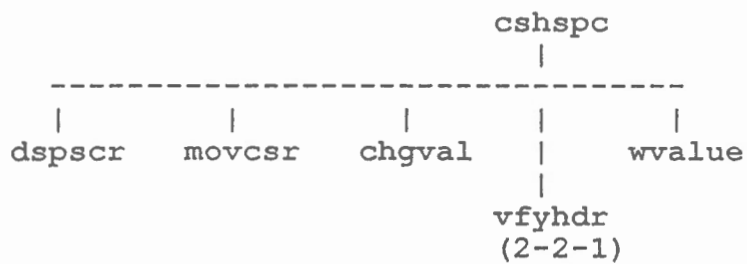


Figure 2-2



Figure 2-2-1

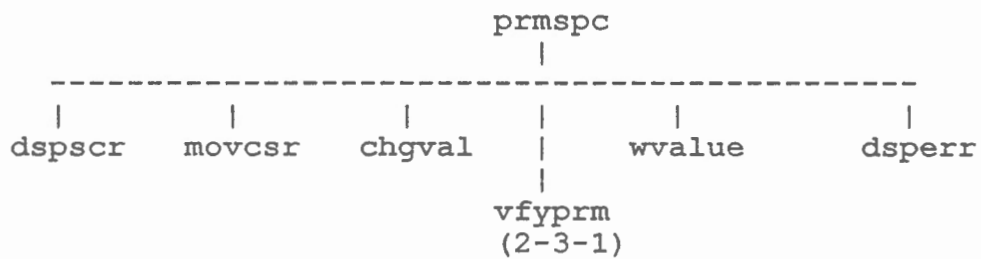


Figure 2-3

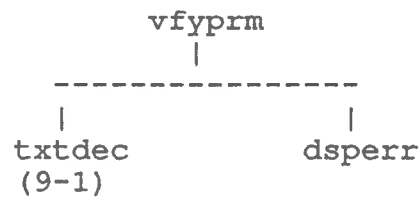


Figure 2-3-1

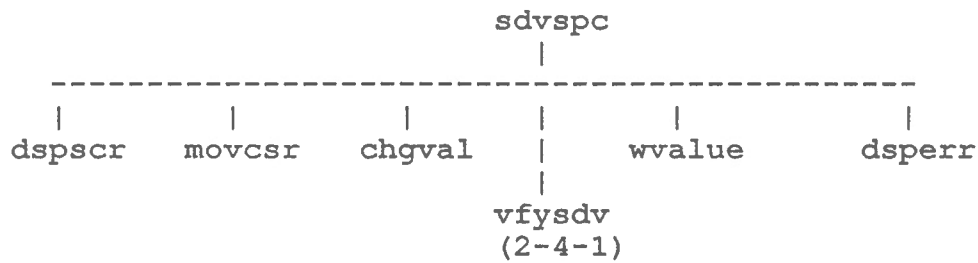


Figure 2-4

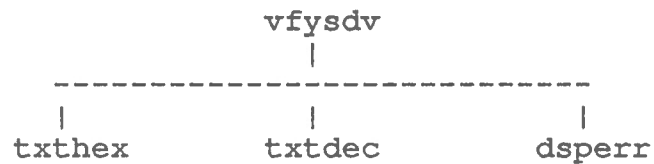


Figure 2-4-1

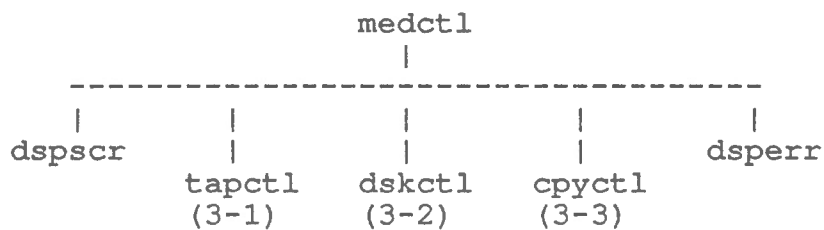


Figure 3-0



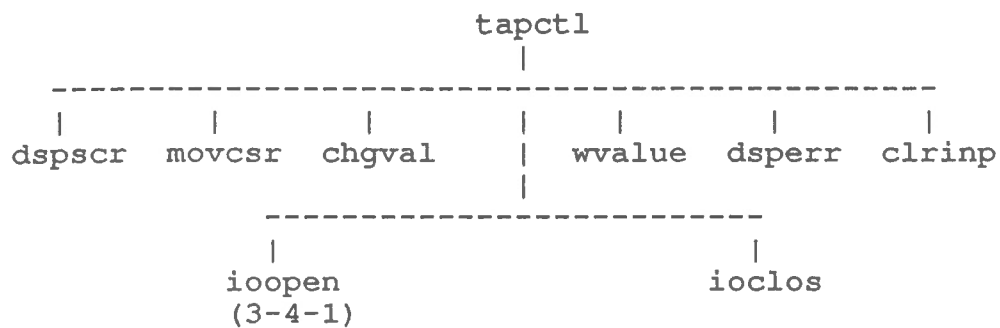


Figure 3-1

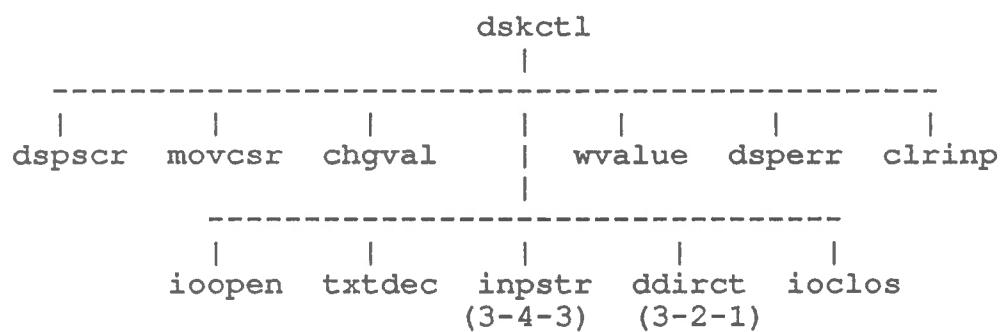


Figure 3-2

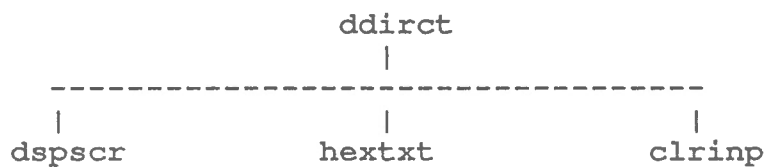


Figure 3-2-1



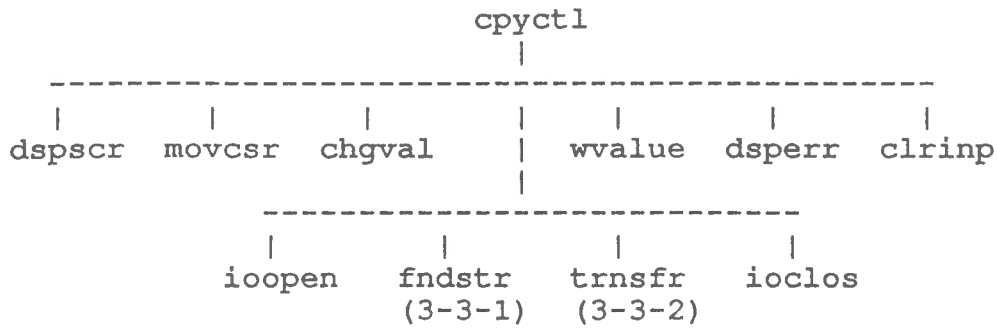


Figure 3-3

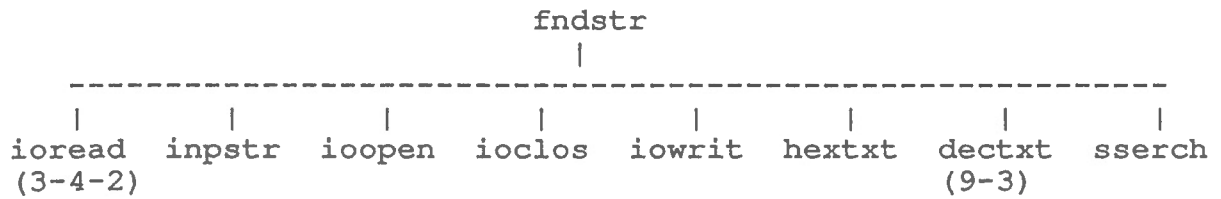


Figure 3-3-1

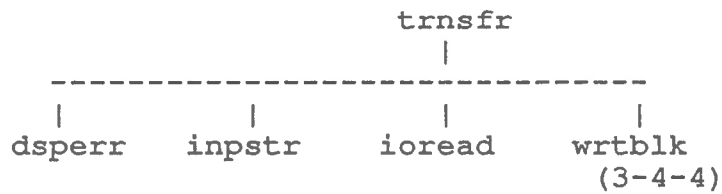


Figure 3-3-2



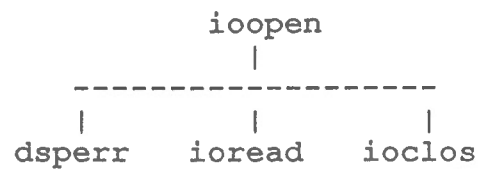


Figure 3-4-1



Figure 3-4-2



Figure 3-4-3

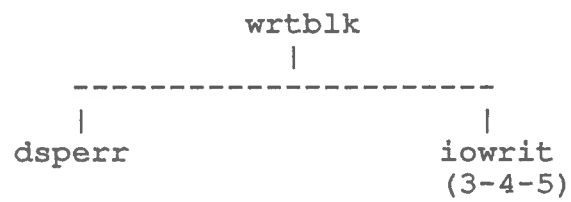


Figure 3-4-4



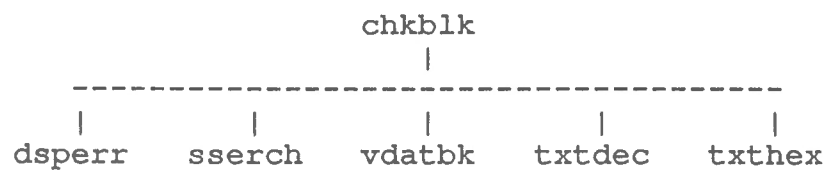


Figure 4-3

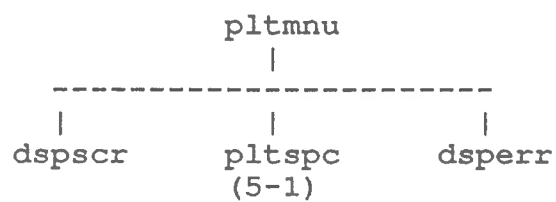


Figure 5-0

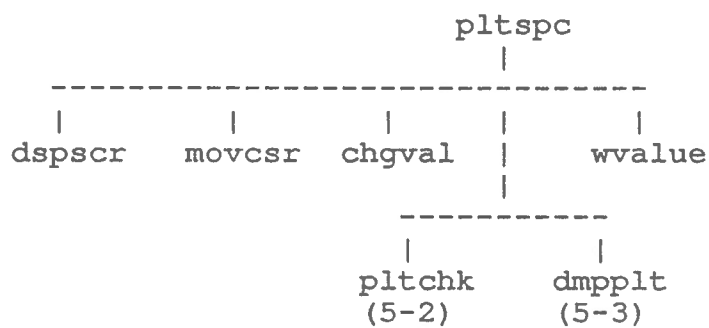


Figure 5-1

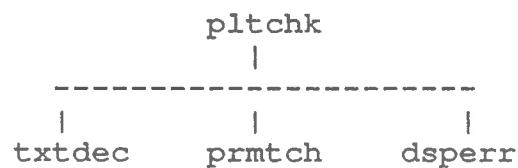


Figure 5-2



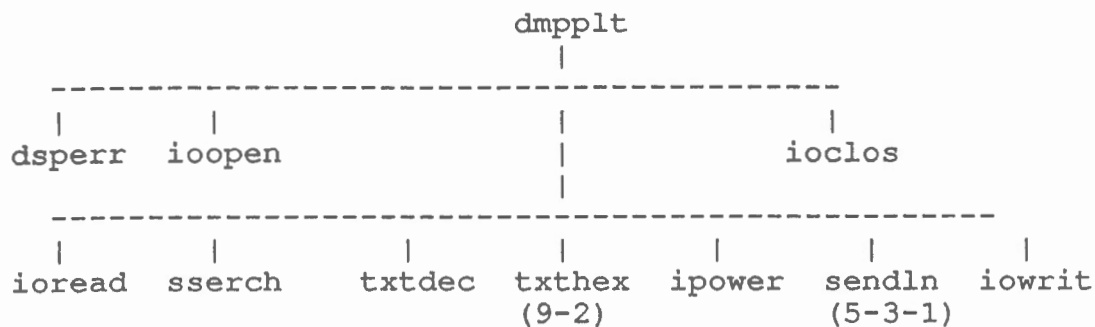


Figure 5-3



Figure 5-3-1

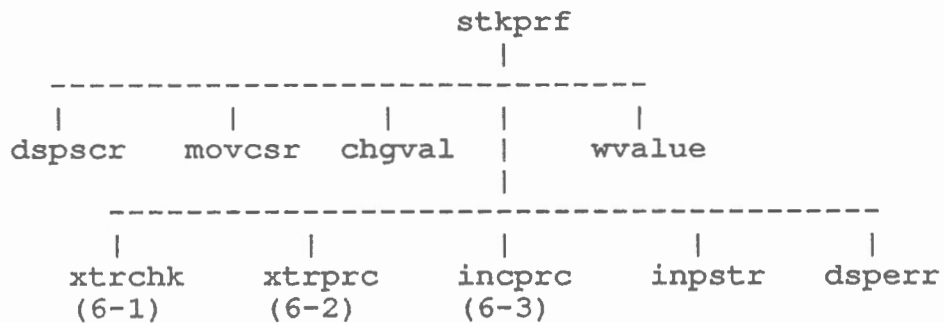


Figure 6-0



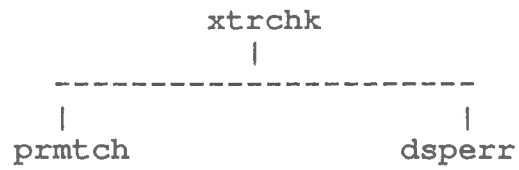


Figure 6-1

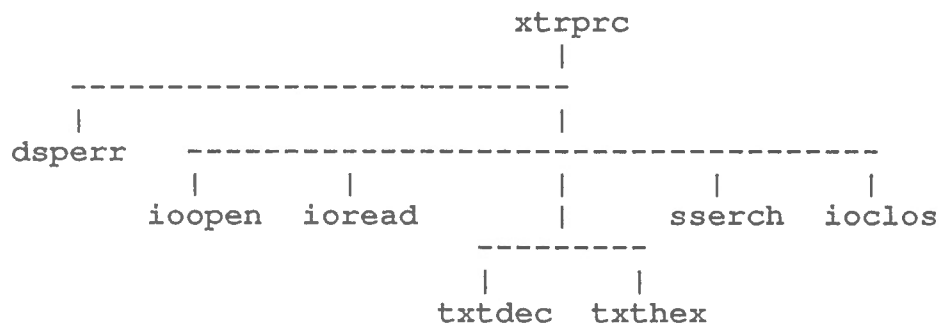


Figure 6-2

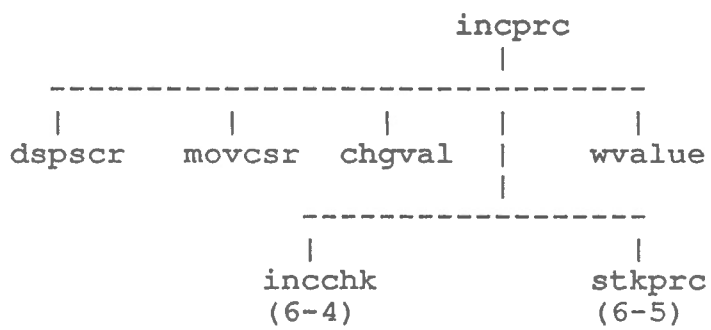


Figure 6-3



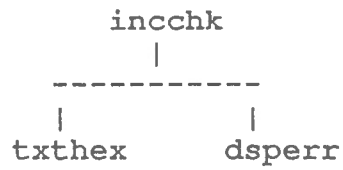


Figure 6-4

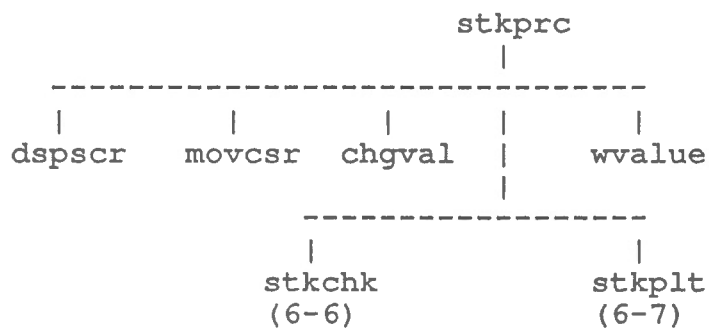


Figure 6-5

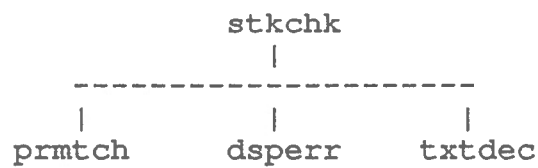


Figure 6-6



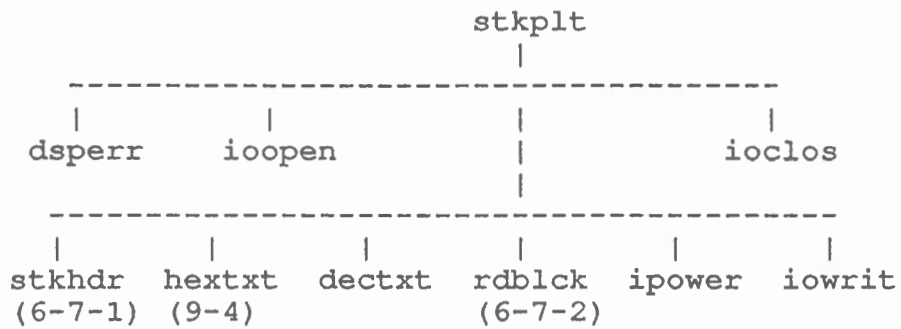


Figure 6-7

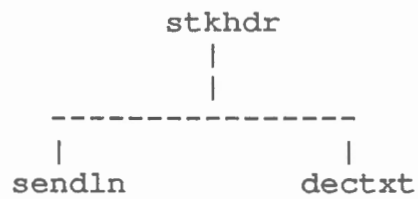


Figure 6-7-1



Figure 6-7-2



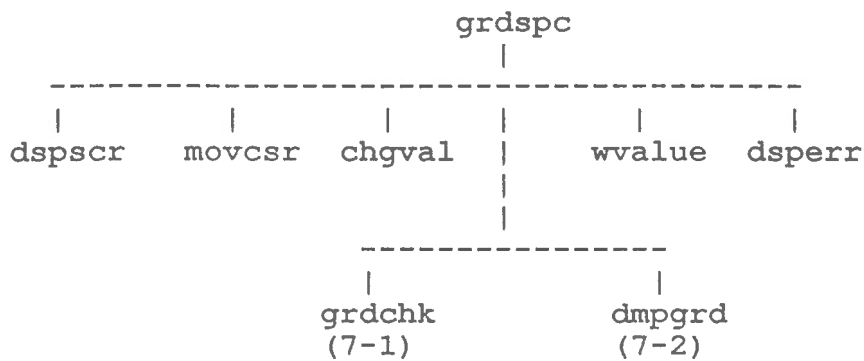


Figure 7-0

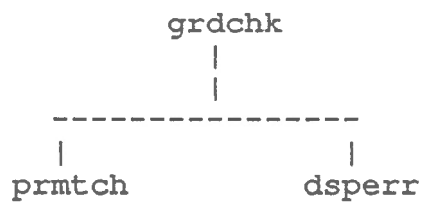


Figure 7-1

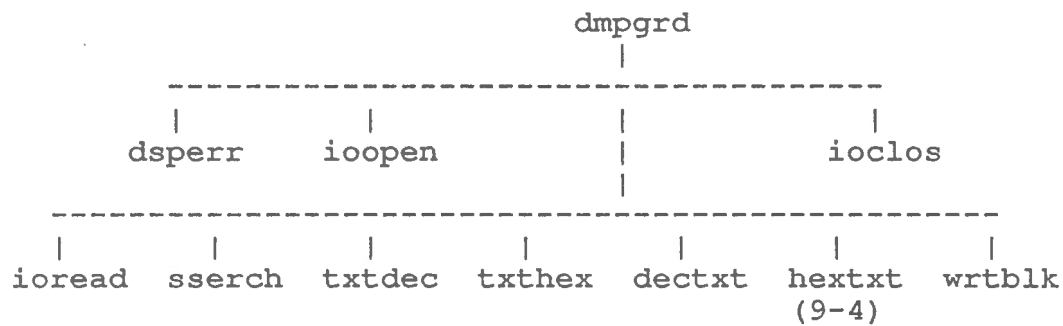


Figure 7-2



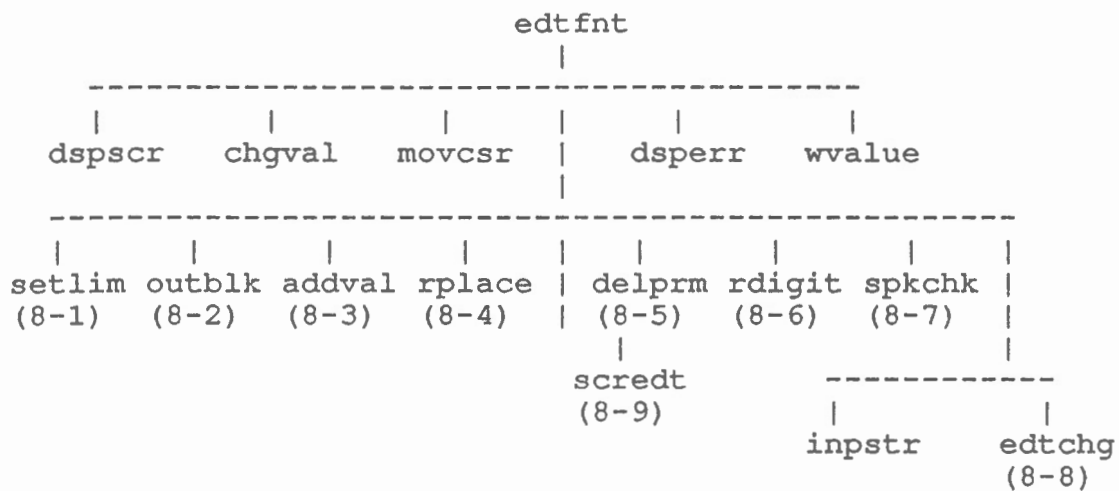


Figure 8-0



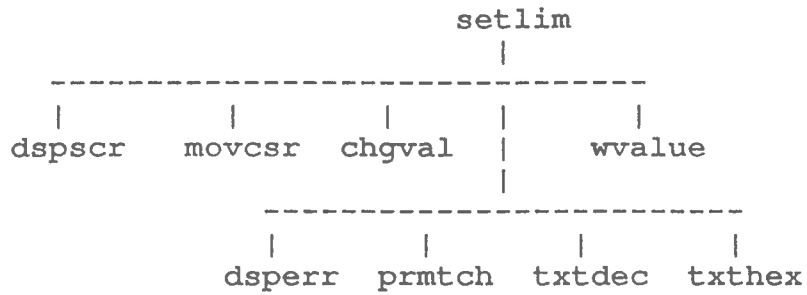


Figure 8-1



Figure 8-2

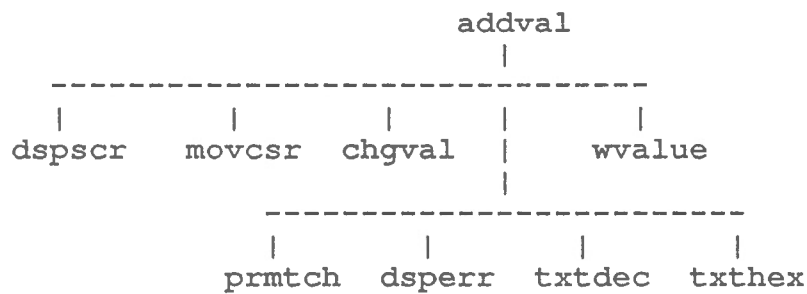


Figure 8-3





Figure 8-4

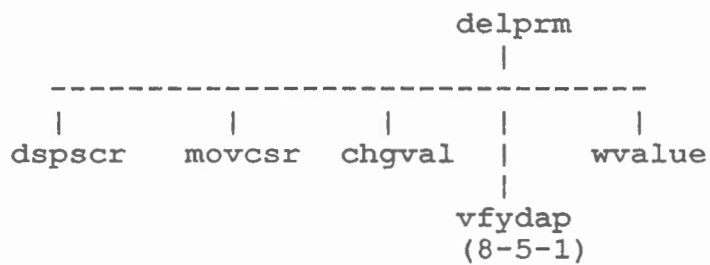


Figure 8.5

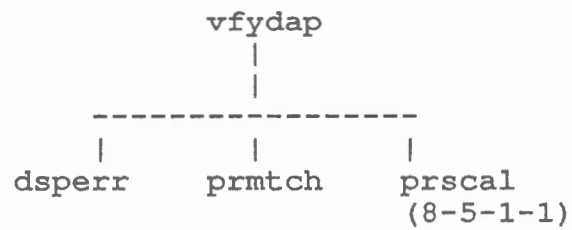


Figure 8-5-1



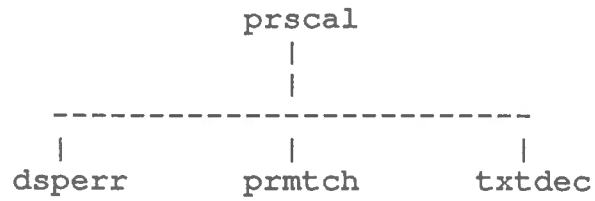


Figure 8-5-1-1

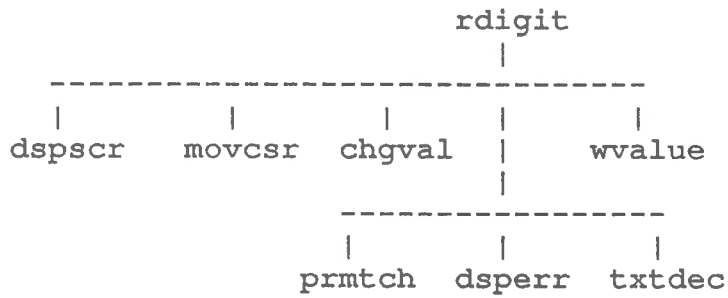


Figure 8-6

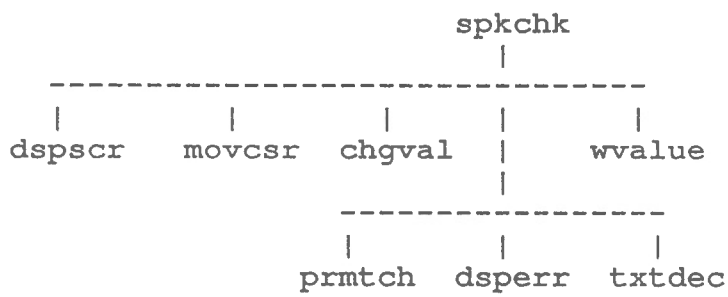


Figure 8-7



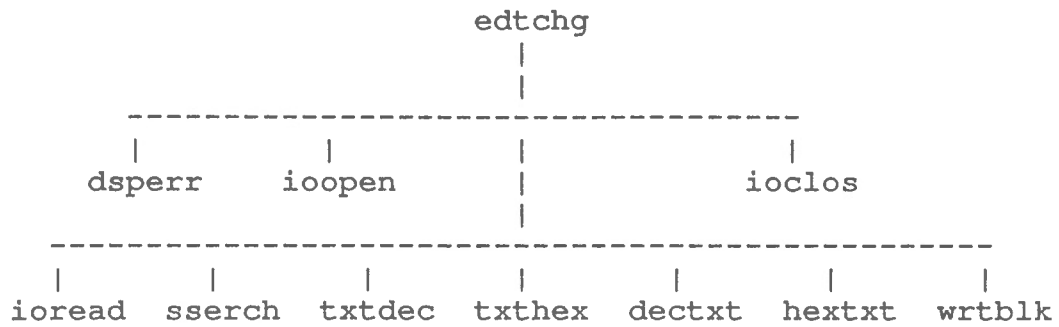


Figure 8-8

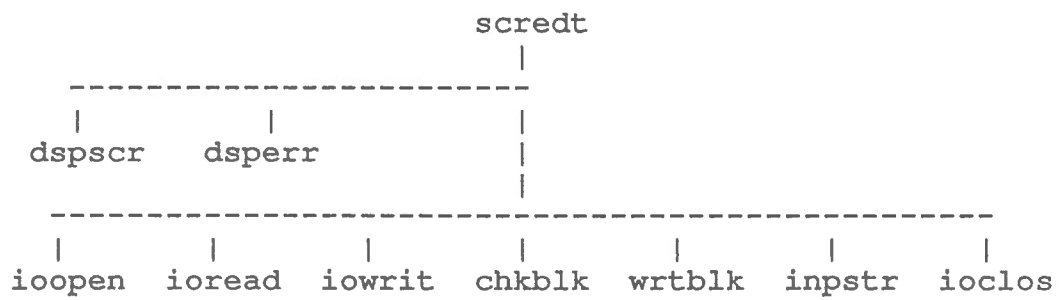


Figure 8-9



```
txtdec
|
dsperr
```

Figure 9-1

```
txthex
|
dsperr
```

Figure 9-2

```
dectxt
|
dsperr
```

Figure 9-3

```
hextxt
|
dsperr
```

Figure 9-4



SECTION 3

PROGRAM STRUCTURE

	page
SCOPE	
1.0 Mainline and Display Routines	1
2.0 System Specifications Routines	33
3.0 Medium Control and Input/Output Routines	55
4.0 Block Printing Routines	93
5.0 Plotting Routines	110
6.0 Stacked Profiles Routines	132
7.0 Gradient Parameter Creation Routines	164
8.0 Editing Routines	173
9.0 Text/Numeric Conversion Routines	220



#2100-12-002.01.0

SCOPE

The Field Checking System is comprised of several routines. This section of the Design Document will provide the details for each of the routines in the System. Each description will include the name and function of the routine, a synopsis, input and output parameters, return values, other functions and global variables referenced, and an algorithm. The algorithm should provide sufficient detail such that conversion to a C language procedure may be quick and straightforward.



2100-12-002

1.0 MAINLINE AND DISPLAY ROUTINES

1.1 Mainline

```

/*****
*
*   Name:
*       main
*
*   Function:
*       Process Operator choices from Main Menu
*
*   Synopsis:
*       fcs [-p]
*
*   Input Parameters:
*       -p : print screen messages to Error Console
*   Output Parameters:
*
*   Return Values:
*
*   Functions Referenced:
*       initial:    initialize tables and screen windows
*       dsperr:     display error text on screen
*       dspscr:     places menu/display and prompt on screen
*       edtfnt:     edit blocks of data
*       endwin:     finish up screens
*       exit:       abort program execution
*       fmtspc:     print data blocks
*       grdspc:     define gradient specifications
*       medctl:     manipulate recording media
*       noraw:      unset raw mode
*       pltmnu:     create a plot
*       printf:     print error message on monitor
*       raw:        set terminal to raw mode
*       stkprf:     create a stacked profile
*       strcmp:     compare two strings for equality
*       syspec:     define system specifications
*       wgetch:     get a character through a window
*
*   Global Variables Referenced:
*       input_line: Input Line window of display
*       msg_prt:    set message console print flag based on invocation of fcs
*****/
```

2100-12-002

```
*
* Description:
*   Initializes Field Checking System, displays Main Menu, prompts Operator
*   for option and processes option until Operator quits.
*
* Algorithm:
*
*   begin
*     if only one invocation argument then
*       set message print flag to FALSE
*     else if two arguments and [strcmp] shows second argument is "-p" then
*       set message print flag to TRUE
*     else
*       begin
*         call [printf] with "fcs: invalid option"
*         call [exit] to abort field checking system
*       end
*     call [initall] to initialize tables and display windows
*     if error from [initall] then
*       call [exit] to abort field checking system
*     call [raw] to unset raw mode
*     set input response to 0
*     while input response not CTRL Z do
*       begin
*         call [dspscr] to display Main Menu
*         if error from [dspscr] then
*           exit
*         call [wgetch] to get Operator choice
*         case Operator choice of
*           1 : call [syspec] to define system specifications
*           2 : call [medctl] to control media
*           3 : call [fmtspc] to print data blocks
*           4 : call [pltmnu] to perform plot
*           5 : call [stkprf] to create a stacked profile
*           6 : call [grdspc] to create gradient parameters
*           7 : call [edtfnt] to perform edit functions
*           CTRL Z: no operation
*           default: call [dsperr] to display message "option not available"
*         end
*       call [endwin] to finish up screens
*       call [noraw] to set terminal to noraw mode
*       call [printf] to backup cursor to beginning of line on exit
*     end
*
*****/
```

1.2 Initialize System

```

/*****
*
* Name:
*   initial
*
* Function:
*   Initialize global variables
*
* Synopsis:
*   initial()
*
* Input Parameters:
*
* Output Parameters:
*   air_chk_set:    airborne data character-checking set
*   diu_chk_set:    diurnal data character-checking set
*   lun_tbl:        Logical Unit Numbers Table
*
* Return Values:
*   NULL_ERR = okay
*   negative = error encountered
*
* Functions Referenced:
*   filerd:    read text from a file into a table
*   fldlnk:    link the fields of a display
*   initscr:    initialize screens
*   leaveok:    set leave flag for window
*   refresh:    make current screen on display
*   rvalue:    read the value file for a given display
*   scrollok:    set scroll flag
*   subwin:    create window within window
*   vfyhdr:    verify data character-checking set and header
*   vfylun:    verify logical unit number assignments
*   vfyprm:    verify recording parameters
*   vfysdv:    verify storage device specifications
*   wclear:    clear window
*
* Global Variables Referenced:
*   air_chk_set:    airborne data character-checking set
*   diu_chk_set:    diurnal data character-checking set
*   error_line:    error line window of display
*   error_table:    error text messages

```



```

*   input_line:   input line window of display
*   present_area: presentation area window of display
*   prmt_line:    prompt line window of display
*   prmt_table:   prompt text messages
*   rec_prm:      recording parameters and specifications
*

```

* Description:

```

*   Initializes the Error and Prompt Tables and defines the subwindows of
*   the IBM-AT Monitor, initializes Recording Parameter Table, LUN Table,
*   storage device specifications and character-checking sets for
*   Diurnal and Airborne data.
*

```

* Algorithm:

```

*   begin
*       set return value to NULL_ERR
*       call [filerd] to set up error_table
*       if no error from [filerd] then
*           call [filerd] to set up prmt_table
*       if no errors from [filerd] then
*           begin
*               call [initscr] to initialize screens
*               call [subwin] to define error_line
*               call [wclear] to clear error_line
*               call [leaveok] with TRUE for error_line
*               call [scrollok] with FALSE for error_line
*               call [subwin] to define input_line
*               call [wclear] to clear input_line
*               call [leaveok] with TRUE for input_line
*               call [scrollok] with FALSE for input_line
*               call [subwin] to define prmt_line
*               call [wclear] to clear prmt_line
*               call [leaveok] with TRUE for prmt_line
*               call [scrollok] with FALSE for prmt_line
*               call [subwin] to define present_area
*               call [wclear] to clear present_area
*               call [leaveok] with TRUE for present_area
*               call [scrollok] with FALSE for present_area
*               call [refresh] with stdscr to clear display
*           end
*       if no errors then
*           call [rvalue] to read the value file for the Recording Parameters
*
*           Display -- page 1
*       if no errors then
*           begin
*               call [fldlnk] to link the fields
*               call [vfyprm] to verify and load the table
*

```

```

*      end
*      if no errors then
*          call [rvalue] to read the value file for the Recording Parameters
*                                          Display -- page 2
*      if no errors then
*          begin
*              call [fldlnk] to link the fields
*              call [vfyprm] to verify and load the table
*          end
*      if no errors then
*          call [rvalue] to read the value file for the Recording Parameters
*                                          Display -- page 3
*      if no errors then
*          begin
*              call [fldlnk] to link the fields
*              call [vfyprm] to verify and load the table
*          end
*      if no errors then
*          call [rvalue] to read the value file for the Logical Unit Number
*                                          Display
*      if no errors then
*          begin
*              call [fldlnk] to link the fields
*              call [vfylun] to verify and load the table
*          end
*      if no errors then
*          call [rvalue] to read the value file for the Airborne Check Set
*                                          and Header Display
*      if no errors then
*          begin
*              call [fldlnk] to link the fields
*              call [vfyhdr] to verify and load the table
*          end
*      if no errors then
*          call [rvalue] to read the value file for the Diurnal Check Set
*                                          and Header Display
*      if no errors then
*          begin
*              call [fldlnk] to link the fields
*              call [vfyhdr] to verify and load the table
*          end
*      if no errors then
*          call [rvalue] to read the value file for the Storage Device
*                                          Specifications Display
*      if no errors then
*          begin

```

2100-12-002

```
*          call [fldlnk] to link the fields
*          call [vfysdv] to verify and load the table
*          end
*      return status value
*  end
*
*****/
```

1.2.1 Read System Prompts/Errors

```

/*****
*
* Name:
*   filerd
*
* Function:
*   Read a text file
*
* Synopsis:
*   filerd(file,oset,table,txt_str)
*
* Input Parameters:
*   file:      character array of the file name to be read
*   oset:      offset into table to zero index position
*   *table:    contains pointers to strings of text
*   *txt_str:  contains actual text strings
*
* Output Parameters:
*
* Return Values:
*   NULL_ERR = okay
*   GENERAL_ERR = cannot find specified file
*
* Functions Referenced:
*   fclose:   close file for reading
*   fopen:    open a file for reading
*   printf:   print message on a specified device
*   getc:     get a character from a file
*   txtdec:   convert text to decimal value
*
* Global Variables Referenced:
*
* Description:
*   Reads the text of a file and places the text in the table passed.
*
* Algorithm:
*   begin
*       set return value to NULL_ERR
*       call [fopen] to open file
*       if error from [fopen] then
*           begin
*               call [printf] on stdout "fcs: cannot open file"

```

2100-12-002

```
*      set return value to GENERAL_ERR
*      end
*      else
*      begin
*      set first text pointer and max and min codes to 0
*      while not end-of-file do
*      begin
*      call [getc] to get a character from the file
*      while character is not the end-of-line do
*      begin
*      save the character in a temporary array
*      call [getc] to get another character
*      end
*      skip over spaces in temporary array
*      call [txtdec] to extract index value from temporary array
*      if the index is less than the min code then
*      save the index as the min code
*      else if the index is greater than the max code then
*      save the index as the max code
*      skip to first quote (") in temporary array
*      save index to start of text, in table
*      while second quote not encountered in temporary array do
*      copy character from temporary array to table
*      mark end of text in table with null character
*      save index to end of text, in table
*      end
*      call [fclose] to close the file
*      end
*      return status value
*      end
*
*****/
```

1.3 Link Fields of Presentation Area

```

/*****
*
*   Name:
*       fldlnk
*
*   Function:
*       Link fields on screen for cursor movement
*
*   Synopsis:
*       fldlnk()
*
*   Input Parameters:
*
*   Output Parameters:
*
*   Return Values:
*       NULL_ERR = okay
*
*   Functions Referenced:
*
*   Global Variables Referenced:
*       scr:  attributes and descriptions of fields on screen
*
*   Description:
*       Links the fields on the screen for left, right, up and down cursor
*       motion between fields
*
*   Algorithm:
*       begin
*           set return value to NULL_ERR
*           clear an array of flags to be used to indicate linked fields
*           count the number of fields on the screen
*           set first field to be linked as the Input Line
*           for each field of screen do
*               begin
*                   set flag for x coordinate to NULL_PTR
*                   set flag for y coordinate to NULL_PTR
*                   for each field of screen do
*                       begin
*                           if starting y coordinate of field is greater than the
*                           y coordinate flag or, is equal and the starting
*                           x coordinate of field is greater than or equal to the

```

```

*           x coordinate flag, and the field has not already been
*           linked then
*       begin
*           set flag for x coordinate to x coordinate for start of field
*           set flag for y coordinate to y coordinate for start of field
*           set field to link as field just examined
*       end
*   end
*   save as field to right, the index found as link
*   set field linked flag
*   set field to link as field just linked
*   end
*   save last field linked as right-cursor index
*   set linked-field's left link to NULL_PTR to indicate Input Line
*   set up the left links for all fields as the reverse of the right links
*   save last field linked as left-cursor index
*   clear an array of flags to be used to indicate linked fields
*   set first field to link equal to the input line
*   for each field of screen do
*       begin
*           set flag for x coordinate to NULL_PTR
*           set flag for y coordinate to NULL_PTR
*           for each field of screen do
*               begin
*                   if starting x coordinate of field is greater than the
*                   x coordinate flag or, is equal and the starting
*                   y coordinate of field is greater than or equal to the
*                   y coordinate flag, and the field has not already been
*                   linked then
*                       begin
*                           set flag for x coordinate to x coordinate for start of field
*                           set flag for y coordinate to y coordinate for start of field
*                           set field to link as field just examined
*                       end
*                   end
*                   save as field to down, the index found as link
*                   set field linked flag
*                   set field to link as field just linked
*                   end
*                   save last field linked as down-cursor index
*                   set linked-field's up link to NULL_PTR to indicate Input Line
*                   set up the up links for all fields as the reverse of the down links
*                   save last field linked as up-cursor index in entry
*                   return status value
*               end
*           end

```

2100-12-002

*****/

1.4 Display Menu

```

/*****
*
*   Name:
*       dspscr
*
*   Function:
*       present a display/menu on monitor
*
*   Synopsis:
*       dspscr(fileid)
*
*   Input Parameters:
*       fileid:  numeric segment of file name containing display text
*
*   Output Parameters:
*
*   Return Values:
*       NULL_ERR = okay
*       negative = error when reading one of template or value files
*
*   Functions Referenced:
*       abs:      find absolute value of an integer
*       box:      draw a box around a window
*       clrlnp:   clear Input Line window of display
*       echo:     turn on keyboard echoing
*       fldlnk:   link fields for cursor motion
*       mvwaddch: move and add character to window
*       noecho:   turn off keyboard echoing
*       refresh:  make the screen look like stdscr
*       resetty:  reset terminal characteristics
*       rtmplt:   read template file of the text for display and place in window
*       rvalue:   read the values (if any) associated with the display
*       savetty:  save terminal characteristics
*       werase:   erase window
*       wrefresh: make screen look like window
*
*   Global Variables Referenced:
*       present_area: presentation area window of display
*       prmt_line:   prompt line window of display
*       error_line:  error line window of display
*
*   Description:

```

2100-12-002

```
*      Reads the requested template and value files for the display, builds
*      the presentation area and prompt line, and refreshes screen.
*
* Algorithm:
*   begin
*       set return value to NULL_ERR
*       call [savetty] to save current terminal characteristics
*       call [noecho] to disable keyboard echoing
*       call [werase] to clear the presentation area
*       if fileid is negative then
*           set value file id to fileid / -10
*       else
*           call [abs] to set value file id to positive value
*       call [rtmplt] to read the static text for the display from a file
*       if no error from [rtmplt] then
*           begin
*               call [rvalue] with flag set to read values associated with display
*               if no negative error returned then
*                   begin
*                       if status returned is NULL_ERR then
*                           call [fldlnk] to link the fields for cursor motion
*                           call [box] to draw a dashed box around the Presentation Area
*                           call [mvwaddch] to place a '+' in the four corners
*                               of the Presentation Area
*                           call [wrefresh] to make the Presentation Area on the screen
*                           call [werase] to clear the error_line window
*                           call [wrefresh] to make the error_line on the screen
*                           call [wrefresh] to make the prmt_line on the screen
*                           call [clrinp] to clear Input Line window
*                       end
*                   end
*               call [resetty] to save current terminal characteristics
*               return status value
*           end
*       end
*
* *****/
```

1.4.1 Read Template File

```

/*****
*
* Name:
*   rtmplt
*
* Function:
*   Read the template file into Presentation Area
*
* Synopsis:
*   rtmplt(fileid)
*
* Input Parameters:
*   fileid:   coded portion of file name for template file
*
* Output Parameters:
*
* Return Values:
*   NULL_ERR      = okay
*   GENERAL_ERR   = no template file found for the display
*   INV_FILE_DATA_ERR = invalid data in template file
*
* Functions Referenced:
*   dsperr:       display error message on screen
*   fclose:       close file
*   fgets:        get a line of text from a file
*   fopen:        open file for reading
*   mvwaddstr:     position cursor and add a string to window at coordinates
*   sprintf:      write to a string
*   sscanf:       get variables from a string of text
*   werase:       clear window
*
* Global Variables Referenced:
*   present_area: Presentation Area window of display
*   pmt_txt:       text for display prompts
*   prmt_line:     Prompt Line window of display
*   prmt_table:    pointers to text for all display prompts
*
* Description:
*   Reads the template file, places prompt text in Prompt Line, and places
*   text in Presentation Area window
*
* Algorithm:

```


1.4.2 Read Values File

```

/*****
*
* Name:
*   rvalue
*
* Function:
*   Read the value file and place text in Presentation Area if requested
*
* Synopsis:
*   rvalue(fileid,dsp_flg)
*
* Input Parameters:
*   fileid:   coded portion of file name for value file
*   dsp_flg:  if non-zero, then load text into Present. Area (ie. show text)
*
* Output Parameters:
*
* Return Values:
*   NULL_ERR      = okay
*   NO_VALUE_FILE_STS = no value file found for the display
*
* Functions Referenced:
*   fclose:       close file
*   fgets:        get a line of text from a file
*   fopen:        open file for reading
*   mvwaddstr:    move cursor and add a string to window at coordinates
*   sprintf:      write to a string
*   sscanf:       scan a string for variables
*   strncpy:      copy n characters from string 2 to string 1
*
* Global Variables Referenced:
*   present_area: Presentation Area window of display
*   scr:          attributes and descriptions of fields on current screen
*
* Description:
*   Reads the value file, places text in a linear array in 'scr', and adds
*   the values read for the respective fields to the Presentation Area if
*   display flag is set.
*
* Algorithm:
*   begin
*       set return value to NULL_ERR

```

2100-12-002

```
*      call [sprintf] to build filename
*      call [fopen] to open the value file
*      if error from [fopen] then
*          set return value to NO_VALUE_FILE_STS
*      else
*          begin
*              set end X coordinate of 1st field in field_layout to 0 (end marker)
*              set fld_ptr of screen to NULL_PTR (indicating Input Line)
*              while end-of-file not encountered and no errors do
*                  begin
*                      call [fgets] to read a line of text from value file
*                      call [sscanf] to get field coordinates
*                      if not four values found in string then
*                          call [dsperr] with "missing parameter(s)"
*                      else
*                          begin
*                              save in field_layout the next available position of dsp_text
*                              copy value (less " for text) to dsp_text
*                              pad dsp_text with spaces for values not maximum length
*                              save in field_layout the last filled position of dsp_text
*                              set end X coordinate for the next field_layout entry to 0
*                              if display flag set then
*                                  begin
*                                      for start Y coordinate to end Y coordinate do
*                                          begin
*                                              call [strncpy] to copy dsp_text to temp. string
*                                              mark the end of temp. string with a null character
*                                              call [mvwaddstr] to position cursor and
*                                                  add temp. string to Presentation Area
*                                          end
*                                      end
*                                  end
*                                  move to next field of display table
*                              end
*                          end
*                      end
*                  end
*              call [fclose] to close the value file
*          end
*      return status value
*  end
```

*****/

1.4.3 Change a Value in Presentation Area

```

/*****
*
*   Name:
*       chgval
*
*   Function:
*       change a value through screen input
*
*   Synopsis:
*       chgval(key)
*
*   Input Parameters:
*       key:   character value entered at terminal by Operator
*
*   Output Parameters:
*
*   Return Values:
*       NULL_ERR    = okay
*       INP_LINE_STS = at Input Line window of display
*
*   Functions Referenced:
*       addch:      add a character to stdscr
*       getyx:      get the current (y,x) screen coordinates
*       move:       move the visible cursor on the screen
*       printf:     write to terminal
*       refresh:    update the window on the screen
*       wrefresh:   make the screen look like stdscr
*
*   Global Variables Referenced:
*       input_line: Input Line window of screen
*       scr:        attributes and description of fields on current screen
*
*   Description:
*       Adds the character entered by Operator to the screen, and to the array
*       containing the values, as text, read in from file
*
*   Algorithm:
*       begin
*           set return value to NULL_ERR
*           if key is CTRL C or CTRL Z then
*               return status value
*           call [getyx] to get the (y,x) coordinate of cursor

```

2100-12-002

```
*      if at Input Line then
*      begin
*      call [waddch] to add character to window
*      call [wrefresh] to update window
*      set return value to INP_LINE_STS
*      end
*      else
*      begin
*      call [addch] to add character to window
*      if at end of X coordinate of field then
*      if at end of Y coordinate of field then
*      call [move] to place the cursor over the new character
*      else
*      call [move] to place the cursor on the next horizontal line
*      call [refresh] to make change on screen
*      add character to text of display values
*      end
*      return status value
*      end
*
*****/
```


2100-12-002

1.4.4 Move Visible Cursor

```

/*****
*
*   Name:
*       movcsr
*
*   Function:
*       Move visible cursor by Operator control
*
*   Synopsis:
*       movcsr(key)
*
*   Input Parameters:
*       key:      keyboard input character from Operator
*
*   Output Parameters:
*
*   Return Values:
*       NOT_CRSR_STS = not a cursor key
*       NULL_ERR     = okay
*
*   Functions Referenced:
*       addch:      add a character to a window
*       getyx:      get (y,x) coordinates
*       move:       move to (y,x) in window
*       printf:     write output to terminal
*       refresh:    make screen look like stdscr
*       wgetch:     get a character through the window
*
*   Global Variables Referenced:
*       input_line: Input Line window of display
*       present_area: Presentation Area window of display
*       scr:        attributes and descriptions of fields on current screen
*
*   Description:
*       Verifies Operator input, and moves cursor between fields as requested
*
*   Algorithm:
*       begin
*           set return value to NULL_ERR
*           call [getyx] to get cursor coordinates
*           case key of
*               CTRL_Z:

```

2100-12-002

```
*      CTRL_C:
*          return status
*          break;
*      tab forward:
*          if not in input_line window then
*              begin
*                  if the x coordinate is not greater than the end of the field then
*                      begin
*                          call [move] to move right one space
*                          call [refresh] to update the screen
*                      end
*                  else if y coordinate is not greater than end of the field then
*                      begin
*                          call [move] to move down one vertical line
*                          call [refresh] to update the screen
*                      end
*                  else
*                      call [printf] to sound the bell -- at end of field
*                  end
*              break
*      rubout:
*          if x is not greater than end of field then
*              begin
*                  call [addch] to place a space at window location
*                  place a space in the display text
*              end
*          if not in input_line window then
*              if x coordinate is not less than the start of the field then
*                  call [move] to move left one space
*              else if y coordinate is not less than start of the field then
*                  call [move] to move up to next vertical line at end of field
*              else
*                  begin
*                      call [move] to move back to start of field
*                      NOTE: adding a character moves cursor
*                      call [printf] to sound the bell -- at start of field
*                  end
*              call [refresh] to update the screen
*          break
*      carriage return:
*          set fld ptr to indicate the cursor is in the input_line window
*          call [move] to move the visible cursor to the input_line window
*          call [refresh] to present the cursor on the screen
*          break
*      '[':      (square bracket = insert space)
*          if in Input Line then
```

2100-12-002

```
*      call [printf] to sound the bell -- cannot insert
*
* else
*   begin
*     if x coordinate is greater than the end of the field then
*       call [printf] to sound the bell -- cannot insert
*     else
*       begin
*         set k to end of display text for field
*         set i to end of field X coordinate
*         set j to end of field Y coordinate
*         set done flag to FALSE
*         move cursor to (i,j)
*         while not done do
*           begin
*             if (i,j) coordinate is (x,y) coordinate from [getyx] then
*               begin
*                 call [addch] to place a space on the screen
*                 place a space at location k of display text
*                 set done flag to TRUE
*               end
*             else
*               begin
*                 set d_txt[k] to d_txt[k - 1] (move to right)
*                 call [addch] to add d_txt[k] to screen
*                 decrement k
*                 if i = start of field x coordinate and j is greater
*                   than the y coordinate from [getyx] then
*                     begin
*                       set i to end of field x coordinate
*                       decrement j
*                     end
*                 else
*                   decrement i
*                 call [move] to move cursor to new (i,j)
*               end
*             end
*           end
*         call [move] to place cursor at original location
*         call [refresh] to update the screen
*       end
*     end
*   end
*   break
* 'J':      (square bracket = delete character)
*   if in Input Line then
*     call [printf] to sound the bell -- cannot delete
*   else
*     begin
```

```

*      if x coordinate is greater than the end of the field then
*          call [printf] to sound the bell -- cannot delete
*      else
*          begin
*              set k to char. position of display text for cursor location
*                  in field
*              set i to current cursor X coordinate
*              set j to current cursor Y coordinate
*              set done flag to FALSE
*              while not done do
*                  begin
*                      if (i,j) coordinate is end of field coordinates then
*                          begin
*                              call [addch] to place a space on the screen
*                              place a space at location k of d_txt
*                              set done flag to TRUE
*                              end
*                          else
*                              begin
*                                  set d_txt[k] to d_txt[k + 1] (move to left)
*                                  call [addch] to add d_txt[k] to screen
*                                  increment k
*                                  if i = end of field x coordinate and j is less
*                                      than the end of field y coordinate then
*                                      begin
*                                          set i to start of field x coordinate
*                                          increment j
*                                      end
*                                  else
*                                      increment i
*                                  call [move] to move cursor to new (i,j)
*                                  end
*                              end
*                          end
*                      call [move] to return cursor to (x,y) location
*                      call [refresh] to update the screen
*                      end
*                  end
*              break
*      ESC:
*          call [wgetch] to get the second part of the escape character
*          case second key of
*              back tab:
*                  if not in input_line window then
*                      begin
*                          if the x coord is not less than the start of the field then
*                              begin

```

2100-12-002

```
*          call [move] to move left one space
*          call [refresh] to update the screen
*          end
*      else if the y coord not less than start of the field then
*      begin
*          call [move] to move up one line
*          call [refresh] to update the screen
*          end
*      else
*          call [printf] to sound the bell
*      break
*  up arrow:
*      if in input_line window then
*          set cursor movement flag to up field of entry table
*      else
*          set cursor movement flag to up link of current field
*      break
*  down arrow:
*      if in input_line window then
*          set cursor movement flag to down field of entry table
*      else
*          set cursor movement flag to down link of current field
*      break
*  left arrow:
*      if in input_line window then
*          set cursor movement flag to left field of entry table
*      else
*          set cursor movement flag to left link of current field
*      break
*  right arrow:
*      if in input_line window then
*          set cursor movement flag to right field of entry table
*      else
*          set cursor movement flag to right link of current field
*      break
*  otherwise:
*      call [printf] to sound bell
*      return status value
*      break
*  if in the input_line window then
*      call [move] to move the cursor to the input line
*  else
*      call [move] to move to the field indicated by cursor movement
*          flag
*  call [refresh] to move the cursor in the Presentation Area
*  break
```

2100-12-002

```
*      default:
*      set return value to NOT CURSOR STATUS
*      return status value
*      end
*
*****/
```

2100-12-002

1.4.5 Save Presentation Area Field Values

```

/*****
*
*   Name:
*       wvalue
*
*   Function:
*       Write screen values to disk file
*
*   Synopsis:
*       wvalue(fileid)
*
*   Input Parameters:
*       fileid:   coded portion of file name to be written
*
*   Output Parameters:
*
*   Return Values:
*       NULL_ERR   = okay
*       GENERAL_ERR = cannot open file
*
*   Functions Referenced:
*       dsperr:    write error message on monitor
*       fclose:    close file
*       fopen:     open file for writing
*       fputs:     put string in file
*       sprintf:   print to a string
*       strlen:    find length of a string
*
*   Global Variables Referenced:
*       scr:       attributes and descriptions of fields on current screen
*
*   Description:
*       Writes the modified values to the dsp###.val file for the current display
*
*   Algorithm:
*       begin
*           set return value to NULL_ERR
*           call [sprintf] to build filename
*           call [fopen] to open the value file
*           if error from [fopen] then
*               call [dsperr] with GENERAL_ERR
*           else

```

2100-12-002

```
*      begin
*      while not all fields have been saved in file do
*          begin
*              call [sprintf] to save start y,x end y,x coordinates in
*              temporary string and place quote (") at end
*              call [strlen] to find length of temporary string
*              copy field text to temporary string
*              add a quote (") to the temporary string
*              add an end-of-line character to temporary string
*              call [fputs] to move temporary string to value file
*          end
*          call [fclose] to close file
*      end
*      return status value
*  end
*
*****/
```


1.5 Display Error/Status Message

```

/*****
*
* Name:
*     dsperr
*
* Function:
*     Display an error/status message
*
* Synopsis:
*     dsperr(err_code,index)
*
* Input Parameters:
*     err_code:  unsigned index into error table containing text messages
*     index:     field on screen containing error
*
* Output Parameters:
*
* Return Values:
*     err_code = error code of message displayed
*     GENERAL_ERR = err_code not valid table index
*
*     NOTE:  GENERAL_ERR will also be returned if this is the
*           err_code passed
*
* Functions Referenced:
*     ioclos:    close device for input/output
*     ioopen:    open a device for input/output
*     iowrit:    write to a device
*     move:      move cursor on screen
*     refresh:   update screen
*     waddstr:   add string to window
*     werase:    erase window
*     wrefresh:  refresh window of display
*     wstandend: reverse video off
*     wstandout: reverse video on
*
* Global Variables Referenced:
*     error_line: window of display containing error/status messages
*     err_txt:    text for error messages
*     error_table: pointers to text of error/status messages
*     input_line: window of display used for keyboard input
*     lun_tbl:    table mapping physical to logical devices
*     msg_prt:    print display messages to error console flag

```

2100-12-002

```
*      scr:          fields of text and data currently displayed on monitor
*
* Description:
*   The text for the error message is added to the window and presented
*   on the display, and on the printer if requested
*
* Algorithm:
*   begin
*     if error_code is greater than top end of error_table or
*       less than bottom end of error_table then
*       set return value to GENERAL_ERR
*     else
*       begin
*         call [werase] to erase the error window
*         copy the error message to a temporary buffer
*         call [wstandout] to highlight error message
*         call [waddstr] to add text to window
*         call [wrefresh] to refresh Error Line of display
*         call [wstandend] to turn off highlighting
*         call [werase] to erase input window
*         call [wrefresh] to update Input Line of display
*         if screen containing error is NULL_PTR (remain at Input Line) then
*           call [move] to position cursor on Input Line
*         else
*           begin
*             call [move] to position cursor on field containing error
*             set fld_ptr of screen to the field containing the error
*           end
*         call [refresh] to update screen
*       end
*     if printer output is desired and error message is not NULL then
*       begin
*         set printer pointer to NULL_PTR
*         while not entire LUN table searched and printer pointer NULL do
*           if LUN table entry matches star printer then
*             set printer pointer to LUN table index
*         if star printer found in LUN table then
*           begin
*             call [ioopen] to open printer
*             call [iowrit] to print message
*             call [iowrit] to print carriage return
*             call [ioclos] to close printer
*           end
*         end
*       return err_code
*     end
```

2100-12-002

*

*****/

1.6 Clear the Input Line Window

```

/*****
*
* Name:
*   clrinp
*
* Function:
*   clear Input Line Window
*
* Synopsis:
*   clrinp()
*
* Input Parameters:
*
* Output Parameters:
*
* Return Values:
*   NULL_ERR = okay
*
* Functions Referenced:
*   mvwaddstr: move cursor and add character string to window
*   werase:   erase window
*   wmove:    set current (y,x) coordinates on window
*   wrefresh: make screen look like window
*
* Global Variables Referenced:
*   input_line: input line window of display
*
* Description:
*   Clears the Input Line window and positions cursor at start of Input
*   Line Window.
*
* Algorithm:
*   begin
*       set return value to NULL_ERR
*       call [mvwaddstr] to position cursor and add a dummy string to Window
*       call [wrefresh] to make the Input Line on the screen
*       call [werase] to clear the Input Line window
*       call [wmove] to position cursor at beginning of Input Line
*       call [wrefresh] to make the Input Line on the screen
*       return status value
*   end
*
*
*/

```

2100-12-002

*****/

2100-12-002

2.0 SYSTEM SPECIFICATIONS ROUTINES

2.1 Present System Specifications Menu

```

/*****
*
*   Name:
*       syspec
*
*   Function:
*       Make changes to System Specifications
*
*   Synopsis:
*       syspec(fileid)
*
*   Input Parameters:
*       fileid:   coded portion of file name for required display file
*
*   Output Parameters:
*
*   Return Values:
*       NULL_ERR = okay
*
*   Functions Referenced:
*       cshspc:   define data character set and header specifications
*       dsperr:   display error text on screen
*       dspscr:   places menu/display and prompt on screen
*       lunspc:   define logical unit number specifications
*       prmshpc:  define recording parameters
*       sdvspc:   define system storage device specifications
*       wgetch:   get a character through a window
*
*   Global Variables Referenced:
*       air_chk_set:  contains valid airborne data blk chars and print header
*       diu_chk_set:  contains valid diurnal data blk chars and print header
*       input_line:   Input Line window of display
*
*   Description:
*       Presents System Specifications Menu and handles Operator choices until
*       CTRL Z entered
*
*   Algorithm:
*       begin

```

2100-12-002

```
*      set return value to NULL_ERR
*      set input response to 0
*      while input response is not CTRL Z or CTRL C do
*          begin
*              call [dsperr] to display menu
*              if error from [dsperr] then
*                  exit
*              call [wgetch] to get Operator choice
*              case Operator choice of
*                  1 : call [lunspc] to define logical unit numbers
*                  2 : call [prmspc] to define recording parameters
*                  3 : call [cshspc] with airborne checking set
*                  4 : call [cshspc] with diurnal checking set
*                  5 : call [sdvspc] to define system storage device specifications
*                  CTRL Z:
*                  CTRL C:  no operation
*                  default: call [dsperr] to display message "option not available"
*              end
*          return status value
*      end
*
*****/
```

2100-12-002

2.2 Present Logical Unit Specifications

```

/*****
*
*   Name:
*       lunspc
*
*   Function:
*       define logical unit numbers
*
*   Synopsis:
*       lunspc(fileid)
*
*   Input Parameters:
*       fileid:  coded portion of file name for display file
*
*   Output Parameters:
*
*   Return Values:
*       NULL_ERR = okay
*       negative = error encountered
*
*   Functions Referenced:
*       chgval:  change a value on the screen
*       dspscr:  present display on screen
*       echo:    set terminal to echo mode
*       vfylun:  verify LUN assignment defined on Monitor
*       movcsr:  move the cursor as requested
*       noecho:  set terminal to noecho mode
*       wgetch:  get keyboard input from Operator
*       wvalue:  write changed values back to value file
*
*   Global Variables Referenced:
*       input_line: Input Line window of display
*
*   Description:
*       Presents the Logical Unit Numbers Display and allows the Operator to
*       cursor around on the screen, modifying values which are updated in the
*       value file and LUN table
*
*   Algorithm:
*       begin
*           set return value to NULL_ERR
*           call [dspscr] to present display
*
*****/
```


2100-12-002

```
*      if error from [dspscr] then
*          return with status value
*      call [noecho] to set terminal to noecho mode
*      set done flag to FALSE
*      while not done do
*          begin
*              set input response to 0
*              while input response is not CTRL Z or CTRL C do
*                  begin
*                      call [wgetch] to get input key response
*                      call [movcsr] to move cursor if requested
*                      if return value from [movcsr] is NOT_CRSR_STS then
*                          call [chgval] to make change on display
*                      end
*                  if key is CTRL_C then
*                      set done flag to TRUE
*                  else
*                      begin
*                          call [vfylyn] to verify LUNs defined
*                          if return status is NULL_ERR then
*                              begin
*                                  call [wvalue] to save updated values
*                                  set done flag to TRUE
*                              end
*                          end
*                      end
*                  end
*              call [echo] to return terminal to echo mode
*              return status value
*          end
*      end
*****/
```

2100-12-002

2.2.1 Verify Logical Unit Specifications

```

/*****
*
*   Name:
*       vfylun
*
*   Function:
*       verifies logical unit number assignments
*
*   Synopsis:
*       vfylun()
*
*   Input Parameters:
*
*   Output Parameters:
*
*   Return Values:
*       NULL_ERR      = okay
*       INV_DATA_ERR  = invalid character specified for Logical Unit Number
*       DUPL_LUN_ERR  = same LUN was assigned to two or more devices
*
*   Functions Referenced:
*       dsperr:      display error/status message on Error Line
*
*   Global Variables Referenced:
*       lun_tbl:     table mapping logical unit numbers to physical devices
*       scr:         fields and attributes of screen
*
*   Description:
*       Verifies the Logical Unit Numbers set up on the Monitor and saves them
*       in the LUN Table if all are correct.
*
*   Algorithm:
*       begin
*           set return value to NULL_ERR
*           set a temporary buffer the size of LUN table to all 0
*           set screen pointer to "right" field from Input Line window
*           set physical device table entry 0 to 0
*           while screen pointer not at Input Line and no errors do
*               begin
*                   get LUN from display
*                   if LUN > largest available LUN
*                       or LUN < smallest LUN available then

```

2100-12-002

```
*      call [dsperr] to display message "invalid data"
*      else if LUN not zero and flag is set in temp. buffer for LUN then
*      call [dsperr] to display message "duplicate LUN's assigned"
*      else
*      begin
*      set flag in temporary buffer
*      save LUN for physical device in temporary table
*      move to next field on screen
*      end
*      end
*      if no errors then
*      copy new values to LUN table
*      return status value
*      end
*
*****/
```

2.3 Present Recording Parameters Specifications

```

/*****
*
*   Name:
*       prmspc
*
*   Function:
*       define the set of valid recording parameters
*
*   Synopsis:
*       prmspc(fileid)
*
*   Input Parameters:
*       fileid:  coded portion of file name for display file
*
*   Output Parameters:
*
*   Return Values:
*       NULL_ERR = okay
*       negative = error encountered during a subroutine call
*
*   Functions Referenced:
*       chgval:   change a value on the screen
*       dspscr:   present display on screen
*       echo:     set terminal to echo mode
*       movcsr:   move the cursor as requested
*       noecho:   set terminal to no echo mode
*       system:   performs VENIX O/S command
*       vfyprm:   verify that all recording parameters are valid
*       wgetch:   get keyboard input from Operator
*       wvalue:   write changed values back to value file
*
*   Global Variables Referenced:
*       input_line:  Input Line window of display
*       rec_prm:     Recording Parameter table containing codes, lengths etc.
*
*   Description:
*       Presents the Recording Parameters Specifications Display and allows the
*       Operator to cursor around on the screen, modifying values which
*       are updated in the values file and recording parameters table
*
*   Algorithm:
*       begin

```

2100-12-002

```
*      set return value to NULL_ERR
*      call [dspscr] to present display
*      if error from [dspscr] then
*          return status
*      call [noecho] to turn off keyboard echoing
*      set temporary fileid (fid) to current fileid
*      set temporary rec. parm. table to existing rec. parm. table
*      clear page modification flags
*      set input response to 0
*      while input is not ((CTRL Z and no errors) or CTRL C)) do
*          begin
*              call [wgetch] to get input
*              call [movcsr] to move cursor if requested
*              if return value from [movcsr] is NO_CSR_STS or key is CTRL Z then
*                  begin
*                      call [chgval] to make change on display
*                      if return value from [chgval] is INF_LINE_STS or
*                          key is CTRL Z then
*                          begin
*                              if key is not valid page number or CTRL C or CTRL Z then
*                                  call [dsperr] with "option not available"
*                              else if key not CTRL C then
*                                  begin
*                                      call [vfyprm] to validate recording parameters
*                                      if no error returned from [vfyprm] then
*                                          begin
*                                              if key is not CTRL Z and
*                                                  page has not been modified previously then
*                                                  begin
*                                                      call [system] to save existing value file as backup
*                                                      set flag indicating page has now been modified
*                                                      end
*                                                      call [wvalue] to save any modifications
*                                                      if key is a page number then
*                                                          call [dspscr] to present the new page
*                                                      end
*                                                  end
*                                              end
*                                          end
*                                      end
*                                  end
*                              end
*                          end
*                      end
*                  end
*              if key is CTRL Z then
*                  begin
*                      copy new rec. parameter values to rec. parm. table
*                      for each parameter page do
*                          if page has been modified then
*                              call [system] to remove the backup value file version
```

2100-12-002

```
*          end
*      else
*          begin
*              for each page do
*                  if page has been modified then
*                      call [system] to restore the previous value file version
*                  end
*              call [echo] to restore keyboard input echoing
*              return status value
*          end
*
*****/
```

2.3.1 Verify Recording Parameters Specifications

```

/*****
*
*   Name:
*       vfyprm
*
*   Function:
*       Verify/update recording parameter set
*
*   Synopsis:
*       vfyprm(fid,rec_tbl)
*
*   Input Parameters:
*       fid:      display fileid indicating which page is on the Monitor
*       rec_tbl:  temporary version of recording parameter table
*
*   Output Parameters:
*
*   Return Values:
*       NULL_ERR      = okay
*       DUPL_P_NAME_ERR = duplicate parameter names have been defined
*       DUPL_P_CODE_ERR = duplicate parameter codes have been defined
*       INV_P_CODE_ERR  = an invalid parameter code has been defined
*       INV_P_NAME_ERR  = an invalid parameter name has been defined
*       INV_P_LEN_ERR   = an invalid parameter length has been defined
*       INV_P_TYP_ERR   = an invalid parameter type has been defined
*       INV_SIGN_ERR    = invalid sign given to a parameter value
*       LIMIT_ERR       = scale factor either too large or too small
*
*   Functions Referenced:
*       dsperr:  display message on Error Line of monitor
*       txtdec:  converts text to decimal equivalent
*       strncmp: compare two strings to determine if identical
*
*   Global Variables Referenced:
*       scr:     layout of parameter fields as they appear on the display
*
*   Description:
*       Verifies that all parameter names and codes are unique, codes contain
*       only non-hexadecimal characters, that parameter types are valid,
*       and parameter lengths and scales are valid.
*
*   Algorithm:

```

```

*      begin
*      set return value to NULL_ERR
*      set table boundaries based on fid
*      initialize lengths of all entries to 0
*      if first page then
*          initialize NULL entry of temporary parameter table
*      while not all fields verified and no error detected do
*          begin
*              copy screen text for field to temporary buffer
*              if entire field all blanks then
*                  set length field to -1 to indicate no entry
*              else
*                  begin
*                      if parameter code not made up of 'G' to 'Z' characters then
*                          call [dsperr] to display message "invalid parameter code"
*                      else if parameter length is not '0' .. '9' then
*                          call [dsperr] to display message "invalid parameter length"
*                      else if parameter type not 'D' or 'H' then
*                          call [dsperr] to display message "invalid parameter type"
*                      else if call to [strncmp] determines parameter name is all
*                          spaces (ie. missing) then
*                          call [dsperr] to display message "missing parameter name"
*                      else if sign in not one of '+', 'n', or 'e' then
*                          call [dsperr] to display message "invalid sign"
*                      else
*                          begin
*                              call [txtdec] to convert scale
*                              if scale is greater than or equal to maximum length of a
*                                  parameter or less than or equal to negative length
*                                  of a parameter then
*                                  call [dsperr] with "limit error"
*                              end
*                          if no errors then
*                              begin
*                                  for each element of temporary table and while no errors do
*                                      begin
*                                          if current parm code is same as temporary table entry then
*                                              call [dsperr] with message "duplicate parameter code"
*                                          else if call to [strncmp] determines parm name is
*                                              same as temporary table entry then
*                                              call [dsperr] with message "duplicate parameter name"
*                                          else
*                                              move to next element of temporary table
*                                          end
*                                      if no errors then
*                                          save field values in record table

```


2100-12-002

```
*           end
*           end
*           move to next field
*           increment table pointer
*           end
*           return status value
*       end
*
*****/
```

2100-12-002

2.4 Present Character Check Set and Header

```

/*****
*
* Name:
*   cshspc
*
* Function:
*   define data character-checking set & header
*
* Synopsis:
*   cshspc(char_set,fileid)
*
* Input Parameters:
*   char_set:  character-checking set and header to be modified
*   fileid:    coded portion of file name for display file
*
* Output Parameters:
*
* Return Values:
*   NULL_ERR = okay
*
* Functions Referenced:
*   chgval:    change a value on the screen
*   dspscr:    present display on screen
*   echo:      set terminal to echo mode
*   movcsr:    move the cursor as requested
*   noecho:    set terminal to noecho mode
*   vfyhdr:    verify that all parameters of header are valid
*   wgetch:    get keyboard input from Operator
*   wvalue:    write changed values back to value file
*
* Global Variables Referenced:
*   input_line: input line window of display
*
* Description:
*   Presents the specified Data Character Set Display and allows the
*   Operator to cursor around on the screen, modifying values which
*   are updated in the values file, specified data character set and
*   block header
*
* Algorithm:
*   begin
*       set return value to NULL_ERR

```

2100-12-002

```
*      call [dspscr] to present display
*      if error from [dspscr] then
*          return
*      call [noecho] to set terminal to noecho mode
*      set quit flag to FALSE
*      while not quit do
*          begin
*              set input response to 0
*              while input response not CTRL Z or CTRL C do
*                  begin
*                      call [wgetch] to get input
*                      if input not CTRL Z or CTRL C then
*                          begin
*                              call [movcsr] to move cursor if requested
*                              if return value from [movcsr] is NOT_CRSR_STS then
*                                  call [chgval] to make change on display
*                              end
*                          end
*                      end
*                  if key is CTRL C then
*                      set quit flag to TRUE
*                  else
*                      begin
*                          call [vfyhdr] to validate the header mnemonics & character-
*                              checking set and to update the char_set structure
*                          if no error from [vfyhdr] then
*                              begin
*                                  call [wvalue] to save updated values
*                                  set quit flag to TRUE
*                              end
*                          end
*                      end
*                  end
*              call [echo] to return terminal to echo mode
*              return status value
*          end
*      end
*****/
```

2100-12-002

2.4.1 Verify Character Check Set and Header

```

/*****
*
*   Name:
*       vfyhdr
*
*   Function:
*       Verify Data Character Set and Header
*
*   Synopsis:
*       vfyhdr(data_set)
*
*   Input Parameters:
*
*   Output Parameters:
*       data_set:  verified character-checking set and header
*
*   Return Values:
*       NULL_ERR      = okay
*       DUPL_CHAR_ERR = duplicate characters in char-checking set
*       INV_P_NAME_ERR = invalid parameter name in block header
*       NO_FARM_ERR    = missing parameter
*
*   Functions Referenced:
*       dsperr:  display message on Error Line of monitor
*       strncmp: compares two strings for character matches
*
*   Global Variables Referenced:
*       rec_prm: recording parameter table of codes and names
*       scr:     descriptions and attributes of data on display
*
*   Description:
*       Verifies that characters in the Character Checking Set are unique and
*       all parameter names in the Block Header exist in the rec_prm
*       table
*
*   Algorithm:
*       begin
*           set return value to NULL_ERR
*           / verify check-set /
*           initialize a temporary data check set to all zeroes
*           find the last character of the set on the screen (chr_end)
*           set check character pointer to start of field

```

```

*       while not at end of characters of field and set is not full
*           and no errors do
*           if entry in check set at ordinal of char being checked then
*               call [dsperr] to display message "duplicate characters found"
*           else
*               set a flag for character in check set at ordinal of char.
*       if no errors then
*           begin
*               / verify header /
*               find end of block header by ignoring spaces at end of field
*               set rec. parm. tbl. ptr. (rptp) to 0
*               while no errors and not at end of text for header do
*                   begin
*                       set space pointer to current character pointer
*                       skip spaces in field to next mnemonic
*                       set # of preceding spaces to new cur. char. ptr - space pointer
*                       copy the mnemonic to a temporary string
*                       pad end with spaces if necessary
*                       set found flag to FALSE
*                       while not found and no errors do
*                           begin
*                               if a test mnemonic string from recording parm table exists for
*                               the entry (length is +ve) then
*                                   call [strncmp] to compare temporary string and test string
*                                   if return value from [strncmp] is not equal to 0 then
*                                       if last element of recording parameter table then
*                                           call [dsperr] with message "invalid parameter name"
*                                       else
*                                           move to next element of recording parameter table
*                                   end
*                               if no errors (ie. match was found) then
*                                   begin
*                                       save parm. table index in temporary data set at rptp
*                                       save number of preceding spaces in temp. data set at rptp
*                                       increment rptp pointer
*                                   end
*                               end
*                           end
*                       end
*                   end
*               if no errors then
*                   begin
*                       if last parameter found was not NULL parameter (see NOTE 1) then
*                           call [dsperr] with "missing parameter"
*                       else
*                           begin
*                               copy data check set table to return parameter
*                               copy header to return parameter

```

2100-12-002

```
*           mark the end of the header table with the NULL pointer
*           end
*           end
*           return status value
*       end
*
*       NOTE 1: last parameter in header must be the NULL parameter
*               to mark the last column of the header
*
*****/
```

2100-12-002

2.5 Present Storage Device Specifications

```

/*****
*
*   Name:
*       sdvspc
*
*   Function:
*       define storage device specifications
*
*   Synopsis:
*       sdvspc(fileid)
*
*   Input Parameters:
*       fileid:  coded portion of file name for display file
*
*   Output Parameters:
*
*   Return Values:
*       NULL_ERR = okay
*       negative = error encountered
*
*   Functions Referenced:
*       chgval:   change a value on the screen
*       dspscr:   present display on screen
*       echo:     set terminal to echo mode
*       vfysdv:   verify storage device specifications defined on Monitor
*       movcsr:   move the cursor as requested
*       noecho:   set terminal to noecho mode
*       wgetch:   get keyboard input from Operator
*       wvalue:   write changed values back to value file
*
*   Global Variables Referenced:
*       input_line: Input Line window of display
*
*   Description:
*       Presents the Storage Device Specifications Display and allows the
*       Operator to cursor around on the screen, modifying values which are
*       updated in the value file and Device Specifications table
*
*   Algorithm:
*       begin
*           set return value to NULL_ERR
*           call [dspscr] to present display

```

2100-12-002

```
*      if error from [dspscr] then
*          return with status value
*      call [noecho] to set terminal to noecho mode
*      set done flag to FALSE
*      while not done do
*          begin
*              set input response to 0
*              while input response is not CTRL Z or CTRL C do
*                  begin
*                      call [wgetch] to get input key response
*                      call [movcsr] to move cursor if requested
*                      if return value from [movcsr] is NOT_CRSR_STS then
*                          call [chgval] to make change on display
*                      end
*                  if key is CTRL_C then
*                      set done flag to TRUE
*                  else
*                      begin
*                          call [vfysdv] to verify specifications defined
*                          if return status is NULL_ERR then
*                              begin
*                                  call [wvalue] to save updated values
*                                  set done flag to TRUE
*                              end
*                          end
*                      end
*                  end
*              call [echo] to return terminal to echo mode
*              return status value
*          end
*      end
*****/
```


2.5.1 Verify Storage Device Specifications

```

/*****
*
*   Name:
*       vfysdv
*
*   Function:
*       verifies storage device specifications
*
*   Synopsis:
*       vfysdv()
*
*   Input Parameters:
*
*   Output Parameters:
*
*   Return Values:
*       NULL_ERR = okay
*       negative = error encountered
*
*   Functions Referenced:
*       dsperr:   display error/status message on Error Line
*       txtdec:   convert text to decimal integer value
*       txthex:   convert text to hexadecimal integer value
*
*   Global Variables Referenced:
*       d_map0:   disk directory for Iomega drive #0
*       d_map1:   disk directory for Iomega drive #1
*       scr:      fields and attributes of screen
*       sdv_spc:  storage device specifications
*
*   Description:
*       Verifies the storage device specifications set up on the Monitor and
*       saves them in the storage device specifications table if all are correct.
*
*   Algorithm:
*       begin
*           set return value to NULL_ERR
*           set field pointer to right field from Input Line
*           set temp. storage pointer to 0
*           while not done four times (two bounds for each disk) and no errors do
*               begin
*                   call [txthex] to convert start logical address to integer

```

```

*      if no error from [txthex] then
*      begin
*      if converted value is out of range of valid disk address then
*      call [dsperr] with message "value out of range"
*      else
*      begin
*      save converted value in temp. storage
*      if specification is an ending LAD then
*      if starting LAD is greater than ending LAD then
*      call [dsperr] with message "limit error"
*      move to next field to right
*      increment temp. storage pointer
*      end
*      end
*      end
*      while no errors and not done twice do
*      begin
*      call [txtdec] to convert recording density to integer
*      if no errors then
*      if value is not in range then
*      call [dsperr] with "value out of range"
*      else
*      begin
*      save converted value in temp. storage
*      move to next field to right
*      end
*      if no errors then
*      begin
*      call [txtdec] to convert length useable tape integer
*      if no errors then
*      if value is not in range then
*      call [dsperr] with "value out of range"
*      else
*      begin
*      save converted value in temp. storage
*      move to next field to right
*      end
*      end
*      if no errors then
*      begin
*      call [txtdec] to convert interrecord gap to integer
*      if no errors then
*      if value is not in range then
*      call [dsperr] with "value out of range"
*      else
*      begin

```

2100-12-002

```
*           save converted value in temp. storage
*           move to next field to right
*           end
*         end
*       end
*     if no errors then
*       copy temporary values to storage device table and disk directories
*     return status value
*   end
*
```

*****/

2100-12-002

3.0 MEDIUM CONTROL AND INPUT/OUTPUT ROUTINES

3.1 Present Medium Control Menu

```

/*****
*
* Name:
*   medctl
*
* Function:
*   Present and choose an option for the Medium Control Functions
*
* Synopsis:
*   medctl(fileid)
*
* Input Parameters:
*   fileid:   coded portion of file name for required display file
*
* Output Parameters:
*
* Return Values:
*   NULL_ERR = okay
*
* Functions Referenced:
*   cpyctl:   perform a copy or search on the FCS media
*   dskctl:   present disk control function menu
*   dsperr:   display error text on screen
*   dspscr:   places menu/display and prompt on screen
*   tapctl:   present tape control function menu
*   wgetch:   get a character through a window
*
* Global Variables Referenced:
*   input_line:   Input Line window of display
*
* Description:
*   Presents Medium Control Utility Menu and handles Operator choices until
*   CTRL Z entered
*
* Algorithm:
*   begin
*       set return value to NULL_ERR
*       set input response to 0
*       while input response is not CTRL Z do

```

2100-12-002

```
*      begin
*      call [dsperr] to display menu
*      if error from [dsperr] then
*          return status
*      call [wgetch] to get Operator choice
*      case Operator choice of
*          1 : call [tapctl] to perform a manipulation of magnetic tape
*          2 : call [dskctl] to perform a manipulation of the hard disks
*          3 : call [cpyctl] to transfer data between or search devices
*          CTRL Z: no operation
*          default: call [dsperr] to display message "option not available"
*      end
*      return status value
*  end
*
```

*****/

3.1.1 Tape Control Function

```

/*****
*
*   Name:
*       tapctl
*
*   Function:
*       Performs Magnetic Tape Control Functions
*
*   Synopsis:
*       tapctl(fileid)
*
*   Input Parameters:
*       fileid:  coded portion of file name for display file
*
*   Output Parameters:
*
*   Return Values:
*       NULL_ERR = okay
*       negative = error encountered
*
*   Functions Referenced:
*       chgval:   change a value on the screen
*       clrinp:   clear Input Line window of display
*       dsperr:   display a message on the Error Line window
*       dspscr:   display the Presentation Area and Prompt Line windows on screen
*       echo:     enable keyboard input echoing
*       inpstr:   get a string from the keyboard
*       ioclos:   close a device for access
*       ioctl:    perform low function on a device
*       ioopen:   open a device for access
*       movcsr:   move the visible cursor on the screen
*       noecho:   disable keyboard input echoing
*       wgetch:   get a character from keyboard through window
*       wvalue:   write values file for a given display
*
*   Global Variables Referenced:
*       input_line:  input line window of display
*       lun_tbl:     maps logical unit numbers to physical devices
*       scr:         attributes and descriptions of fields on screen
*
*   Description:
*       Presents Tape Control Function Menu and allows Operator to perform the

```

```

*      various functions via keyboard input
*
* Algorithm:
*   begin
*       set return value to NULL_ERR
*       call [dspscr] to display menu
*       if error from [dspscr] then
*           exit
*       set input response to 0
*       call [noecho] to turn off keyboard echoing
*       set current and old LUN's to 0
*       while input not ((CTRL Z and no error) or CTRL C)
*           begin
*               call [wgetch] to get input response
*               call [movcsr] to move visible cursor
*               if status from [movcsr] is NO_CRSR_STS or key is CTRL Z then
*                   begin
*                       call [chgval] to change screen value
*                       if status from [chgval] is INF_LINE_STS or key is CTRL Z then
*                           begin
*                               set source to right entry of field layout table
*                               if LUN is not a valid character then
*                                   call [dsperr] with "invalid character"
*                               else if LUN is not one of mag. tape units or null device then
*                                   call [dsperr] with "invalid device"
*                               else
*                                   begin
*                                       if current LUN is not same as old LUN then
*                                           begin
*                                               if old LUN not zero then
*                                                   call [ioclos] to close old LUN
*                                               call [ioopen] to open current LUN
*                                               if error from [ioopen] then
*                                                   set old LUN to 0
*                                               else
*                                                   set old LUN to current LUN
*                                           end
*                                       end
*                                   end
*                               end
*                           end
*                       if no errors then
*                           begin
*                               call [dsperr] to clear error line window
*                               case input response of
*                                   1 : call [ioctl] to forward file on tape unit
*                                   2 : call [ioctl] to backward file on tape unit
*                                   3 : call [ioctl] to forward block on tape unit
*                                   4 : call [ioctl] to backward block on tape unit

```

2100-12-002

```
*           5 : call [echo] to allow keyboard echoing
*           call [inpstr] to get string from keyboard
*           call [noecho] to disable keyboard echoing
*           if response is 'y' or 'Y' then
*               call [ioctl] to write EDF on tape unit
*           6 : call [ioctl] to rewind on tape unit
*           CTRL C:
*           CTRL Z:
*               break
*           default: call [dsperr] with message "invalid option"
*           call [clrinp] to clear Input Line window
*           end
*       end
*   end
*   call [ioclos] to close the device
*   if input response is CTRL Z then
*       call [wvalue] to update value file
*       call [echo] to turn on keyboard echoing
*       return status value
*   end
*
*****/
```


3.1.2 Disk Control Function

```

/*****
*
*   Name:
*       dskctl
*
*   Function:
*       Performs Bernoulli Disk Drive Control Functions
*
*   Synopsis:
*       dskctl(fileid)
*
*   Input Parameters:
*       fileid:  coded portion of file name for display file
*
*   Output Parameters:
*
*   Return Values:
*       NULL_ERR = okay
*       negative = error encountered
*
*   Functions Referenced:
*       chgval:   change a value on the screen
*       clrinp:   clear Input Line window on display
*       ddirct:   display disk directory
*       dsperr:   display a message on the Error Line window
*       dspscr:   display the Presentation Area and Prompt Line windows on screen
*       echo:     enable keyboard input echoing
*       inpstr:   get string from keyboard
*       ioctl:    perform low function on a device
*       ioclos:   close a file for access
*       ioopen:   open a device for access
*       lseek:    position VENIX device pointer at specific address
*       movcsr:   move the visible cursor on the screen
*       noecho:   disable keyboard input echoing
*       txtdec:   convert text to decimal integer
*       wgetch:   get a character from keyboard through window
*       wvalues:  write values file for a given display
*
*   Global Variables Referenced:
*       d_map0:   iomega hard disk # 0 directory
*       d_map1:   iomega hard disk # 1 directory
*       input_line: input line window of display

```

2100-12-002

```
*      lun_tbl:      maps logical unit numbers to physical devices
*      scr:          attributes and descriptions of fields on screen
*
* Description:
*   Presents Disk Control Function Menu and allows Operator to perform the
*   various functions via keyboard input
*
* Algorithm:
*   begin
*       set return value to NULL_ERR
*       call [dspscr] to display menu
*       if error from [dspscr] then
*           return status
*       set input response to 0
*       call [noecho] to turn off keyboard echoing
*       set current and old LUN's to 0
*       while not ((input = CTRL Z and no errors) or CTRL C) do
*           begin
*               call [wgetch] to get input response
*               call [movcsr] to move visible cursor
*               if status from [movcsr] is NO_CRSR_STS or key is CTRL Z then
*                   begin
*                       call [chgval] to change screen value
*                       if status from [chgval] is INP_LINE_STS or key is CTRL Z then
*                           begin
*                               set LUN to right entry of field layout table
*                               if LUN is not valid character then
*                                   call [dsperr] with message "invalid character"
*                               else if LUN not one of disk units or null device then
*                                   call [dsperr] with "invalid device"
*                               else
*                                   begin
*                                       if current LUN not same as old LUN then
*                                           begin
*                                               if old LUN not zero then
*                                                   call [ioclos] to close old LUN
*                                               call [ioopen] to open current LUN
*                                               if error from [ioopen] then
*                                                   set old LUN to 0
*                                               else
*                                                   set old LUN to current LUN
*                                           end
*                                       end
*                                   end
*                               if no errors then
*                                   begin
*                                       call [txtdec] to convert sector interleave
```

```

*           if no errors then
*               if interleave value less than minimum or
*                   greater than maximum then
*                   call [dsperr] with "limit error"
*               else
*                   save value
*               end
*           if no errors then
*               begin
*                   call [dsperr] to clear error line window
*                   case input response of
*                       1 : begin
*                           call [echo] to permit keyboard echo on Monitor
*                           call [inpstr] to verify action to be performed
*                           call [noecho] to disallow keyboard echo
*                           if input is 'Y' or 'y' then
*                               call [ioctl] to format the disk
*                           end
*                       2 : begin
*                           call [echo] to permit keyboard echo on Monitor
*                           call [inpstr] to verify action to be performed
*                           call [noecho] to disallow keyboard echo
*                           if input is 'Y' or 'y' then
*                               begin
*                                   clear DSK_MAP_LEN characters of disk directory
*                                   call [lseek] to position at directory
*                                   call [iowrit] to write to disk map lad
*                               end
*                           end
*                       3 : call [wvalue] to save current screen values
*                           call [ioclos] to close the device
*                           call [ioopen] to reopen and read disk directory
*                           call [ddirct] to display disk directory
*                           call [dspscr] to redisplay Disk Function Menu
*                   CTRL C:
*                   CTRL Z:
*                       break
*                   default: call [dsperr] with message "invalid option"
*                   call [clrinp] to clear Input Line window on display
*               end
*           end
*       end
*   end
*   call [ioclos] to close the device
*   if input response is CTRL Z then
*       call [wvalue] to update value file

```

2100-12-002

```
*      call [echo] to turn on keyboard echoing
*      return status value
*      end
*
*****/
```

2100-12-002

3.1.2.1 Display Disk Directory

```

/*****
*
*   Name:
*       ddirct
*
*   Function:
*       display disk directory
*
*   Synopsis:
*       ddirct(fileid,lun,mptr)
*
*   Input Parameters:
*       fileid:  coded portion of file name for display file
*       lun:     logical unit number of disk device
*       mptr:    pointer to device directory
*
*   Output Parameters:
*
*   Return Values:
*       NULL_ERR = okay
*
*   Functions Referenced:
*       clrinp:   clear Input Line window on display
*       dspscr:   present display on screen
*       hextxt:   convert hex value to ASCII text
*       waddch:   add a character to a screen window
*       wgetch:   get keyboard input from Operator
*       wmove:    position cursor at specific location within a window
*       wrefresh: update the screen window on the Monitor
*
*   Global Variables Referenced:
*       input_line:  Input Line window of display
*       present_area: Presentation Area window of display
*       scr:         Presentation Area attributes and layout
*       rec_prm:     Recording parameters, values, lengths etc.
*
*   Description:
*       Presents the disk directory on the IBM-AT Monitor and allows the Operator
*       to page forward and backward through the directory
*
*   Algorithm:
*       begin

```

```

*      set return value to NULL_ERR
*      call [dspscr] to present display
*      if error from [dspscr] then
*          return with status value
*      set input response to 0
*      set page number to 1
*      call [wmove] and [waddch] to place LUN in presentation area
*      set page pointer to page field of display
*      while keyboard input not CTRL Z or CTRL C do
*          begin
*              call [wmove] and [waddch] to place page number in presentation area
*              if page is greater than 9 then
*                  call [wmove] and [waddch] to place second digit of page number
*                  in presentation area
*              else
*                  call [wmove] and [waddch] to place a space for second digit
*              set field pointer to top right corner of screen
*              set empty directory line flag to FALSE
*              for the required number of lines per directory page do
*                  begin
*                      if display line number is greater than max. size of directory or
*                      the directory entry for display line is blank then
*                          set empty flag to TRUE
*                          call [wmove] and [waddch] to add flight date or spaces
*                          to presentation area
*                          move to next field on screen
*                          if the directory line is not empty then
*                              call [hextxt] to convert traverse line number for line to text
*                              call [wmove] and [waddch] to add line number or spaces
*                              to presentation area
*                              move to next field on screen
*                              if the directory line is not empty then
*                                  call [hextxt] to convert record count to text
*                                  call [wmove] and [waddch] to add record count or spaces
*                                  to presentation area
*                              if the directory line is not empty then
*                                  call [hextxt] to convert starting lad for line to text
*                                  call [wmove] and [waddch] to add starting lad for line or spaces
*                                  to presentation area
*                              move to next field on screen
*                              if the directory line is not empty then
*                                  call [hextxt] to convert ending lad for line to text
*                                  call [wmove] and [waddch] to add ending lad for line or spaces
*                                  to presentation area
*                              move to next field on screen
*                          end
*                  end

```

2100-12-002

```
*      call [wrefresh] to display new directory page
*      call [clrinp] to clear Input Line window
*      call [wgetch] to get keyboard input
*      if key is ESC char then
*          begin
*              call [wgetch] to get second segment of keystroke
*              if key is "page up" then
*                  if next page number is beyond end of directory then
*                      set page number to 1
*                  else
*                      increment page number
*              else if key is "page down" then
*                  if previous page number is 1 then
*                      set page number to largest
*                  else
*                      decrement page number
*              end
*          end
*      end
*      return status value
*  end
*
*****/
```

2100-12-002

3.1.3 Copy/Search Control Function

```

/*****
*
*   Name:
*       cpyctl
*
*   Function:
*       Performs Search and Copy for specified devices
*
*   Synopsis:
*       cpyctl(fileid)
*
*   Input Parameters:
*       fileid:  coded portion of file name for display file
*
*   Output Parameters:
*
*   Return Values:
*       NULL_ERR = okay
*       negative = error encountered
*
*   Functions Referenced:
*       chgval:   change a value on the screen
*       clrinp:   clear Input Line Window on display
*       dsperr:   display a message on the Error Line window
*       dspscr:   display the Presentation Area and Prompt Line windows on screen
*       echo:     enable keyboard input echoing
*       fndstr:   find a string match on device
*       ioclos:   close a file for access
*       ioopen:   open a device for access
*       movcsr:   move the visible cursor on the screen
*       noecho:   disable keyboard input echoing
*       trnsfr:   copy data between two devices
*       wgetch:   get a character from keyboard through window
*       wvalue:   write values file for a given display
*
*   Global Variables Referenced:
*       input_line:  input line window of display
*       lun_tbl:     maps logical unit numbers to physical devices
*       scr:         attributes and descriptions of fields on screen
*
*   Description:
*       Presents Search/Copy Menu and allows Operator to perform the

```



```

*      various functions
*
* Algorithm:
*   begin
*       set return value to NULL_ERR
*       call [dspscr] to display menu
*       if error from [dspscr] then
*           return with status value
*       set input response to 0
*       call [noecho] to turn off keyboard echoing
*       set current and old source LUN's to 0
*       set current and old destination LUN's to 0
*       while input not CTRL Z and CTRL C
*           begin
*               call [wgetch] to get input response
*               call [movcsr] to move visible cursor
*               if status from [movcsr] is NO_CRSR_STS or input is CTRL Z then
*                   begin
*                       call [chgval] to change screen value
*                       if status from [chgval] is INP_LINE_STS or input is CTRL Z then
*                           begin
*                               set source to right entry of field layout table
*                               set destination to right of right entry of field layout table
*                               if source character is not valid then
*                                   call [dsperr] with "invalid character"
*                               else if destination character is not valid then
*                                   call [dsperr] with "invalid character"
*                               if source value = destination value then
*                                   call [dsperr] with "source same as destination device"
*                               else if source not magnetic tape or hard disk then
*                                   call [dsperr] with "invalid device"
*                               else if destination not magnetic tape or hard disk then
*                                   call [dsperr] with "invalid device"
*                               else
*                                   begin
*                                       if current source LUN not same as old source LUN then
*                                           begin
*                                               if old source LUN not zero then
*                                                   call [ioclos] to close old source device
*                                               if current source is same as old destination then
*                                                   call [ioclos] to close old destination device
*                                               call [ioopen] to open current source device
*                                               if error from [ioopen] then
*                                                   set old source LUN to 0
*                                               else
*                                                   set old source LUN to current LUN

```

2100-12-002

```
*
*      end
*      if current dest. LUN not same as old dest. LUN then
*      begin
*          if old dest. LUN not zero and old dest. not same as
*              current source then
*              call [ioclos] to close old device
*          call [ioopen] to open current dest. device
*          if error from [ioopen] then
*              set old dest. LUN to 0
*          else
*              set old dest. LUN to current LUN
*          end
*      end
*      end
*      if no errors then
*      begin
*          call [dsperr] to clear error line window
*          case input response of
*              1 : call [fndstr] to position pointer of source device
*              2 : call [fndstr] to position pointer of dest. device
*              3 : call [fndstr] to find string on source device
*              4 : call [fndstr] to find string on dest. device
*              5 : call [trnsfr] to copy data between devices
*              default: call [dsperr] with message "invalid option"
*          call [clrinp] to clear Input Line window
*      end
*      end
*      end
*      call [ioclos] to close the device
*      call [ioclos] to close the device
*      if input response is CTRL Z then
*          call [wvalue] to update value file
*      call [echo] to turn on keyboard echoing
*      return status value
*      end
*
*
*****/
```

2100-12-002

3.1.3.1 Find String Process

```

/*****
*
*   Name:
*       fndstr
*
*   Function:
*       Searches medium for a string
*
*   Synopsis:
*       fndstr(lun,flag)
*
*   Input Parameters:
*       lun:  logical unit number of source device
*       flag: set   = reposition file pointer to block containing match;
*              clear = leave file pointer as is
*
*   Output Parameters:
*
*   Return Values:
*       NULL_ERR = okay
*       negative = error encountered
*
*   Functions Referenced:
*       asctime:  convert time to ascii
*       dectxt:   convert decimal integer to text
*       echo:     allow keyboard echoing
*       hextxt:   convert hexadecimal integer to text
*       ioclos:   close a device for input/output
*       ioopen:   open a device for input/output
*       ioread:   read a block of data from a device
*       iowrit:   write a block of data to a device
*       inpstr:   get search string from keyboard
*       noecho:   do not allow keyboard echoing
*       sserch:   search a string for a match against a second string
*       time:     get system time
*
*   Global Variables Referenced:
*       buffer1:  data block storage area
*       d_map0:   rigid disk #0 directory
*       d_map1:   rigid disk #1 directory
*       lun_tbl:  maps logical unit numbers to physical devices
*       scr:      screen field values and attributes
*****/
```

2100-12-002

```
*
* Description:
*   Gets search string, and performs read of specified device and
*   comparison until either a match is made or the end of the device is
*   encountered. If a match is made then the file pointer for the device
*   is either repositioned to the block containing the match or left alone
*   as requested
*
* Algorithm:
*   begin
*       set return value to NULL_ERR
*       call [echo] to enable keyboard echoing
*       call [inpstr] to prompt that string on screen is as desired
*       call [noecho] to disable keyboard echoing
*       if response not affirmative then
*           return status value
*       find end of string on screen
*       copy string on screen to buffer
*       set found flag to FALSE
*       set block count to 0
*       set block length to maximum number of characters
*       set printer pointer to -1
*       while printer not found in LUN table and not at end of table do
*           if LUN table entry is star printer then
*               set printer pointer to LUN table index
*           else
*               next move to next LUN table entry
*       if star printer not defined in LUN table then
*           call [dsperr] with "ERROR"
*       else
*           call [ioopen] to open star printer
*       while not found and no errors encountered do
*           begin
*               call [ioread] to get a block of data
*               if no error status from [ioread] then
*                   begin
*                       if first block to be read then
*                           begin
*                               find correct length of block
*                               call [time] to get the system time and date
*                               call [asctime] to convert to ASCII
*                               call [iowrit] to print date and time
*                               call [iowrit] to print "searching for string "
*                               call [iowrit] to print string
*                               call [iowrit] to print " ...\n"
*                           end
*                       end
*                   end
*           end
*       end
```

2100-12-002

```
*      call [sserch] to search data block for string
*      if return status from [sserch] indicates match found then
*      begin
*          call [iowrit] to print "match found in block "
*          call [dectxt] to convert to text
*          call [iowrit] to print block number
*          call [iowrit] to print "at character "
*          call [dectxt] to convert to text
*          call [iowrit] to print character position
*          if searching one of the two hard disks then
*          begin
*              get directory pointer to corresponding disk
*              call [iowrit] to print "\n      Disk LAD = "
*              call [hextxt] to convert LAD to text
*              call [iowrit] to print disk address
*          end
*          call [iowrit] to print carriage return
*          set found flag to TRUE
*      end
*      else
*          increment block counter
*      end
*      end
*      if not found then
*          call [iowrit] to print "no match found\n"
*      else if flag is set then
*          begin
*              if device is one of rigid disks then
*              begin
*                  call [lseek] to back up one block on disk
*                  if no error from [lseek] then
*                      adjust current address pointer in directory
*                  end
*              end
*              else
*                  call [ioctl] to back up a block on the magnetic tape
*              end
*          end
*          return status value
*          call [ioclos] to close Star printer (if necessary)
*      end
*
*****/
```

2100-12-002

3.1.3.2 Copy From Source to Destination

```

/*****
*
*   Name:
*       trnsfr
*
*   Function:
*       Performs transfer (copy) of data between specified devices
*
*   Synopsis:
*       trnsfr(src,dst)
*
*   Input Parameters:
*       src:  LUN for source device of data
*       dst:  LUN for destination device of data
*
*   Output Parameters:
*
*   Return Values:
*       NULL_ERR = okay
*       negative = error encountered
*
*   Functions Referenced:
*       dsperr:  send message to Monitor
*       echo:    permit keyboard echoing
*       inpstr:  read a string from the keyboard
*       ioread:  read a buffer1 of data from a specified device
*       noecho:  do not allow keyboard echoing
*       wrtblk:  write a block of data to output device
*
*   Global Variables Referenced:
*       buffer1:  data block storage area
*
*   Description:
*       Controls copying of data from source device to destination device.
*
*   Algorithm:
*       begin
*           set return value to NULL_ERR
*           call [echo] to allow keyboard echoing
*           call [inpstr] to prompt with "are you sure?"
*           call [noecho] to disable keyboard echoing
*           if response not 'y' and not 'Y' then

```

2100-12-002

```
*      return status value
*      set block length to maximum possible
*      set done flag to FALSE
*      call [dsperr] to print message "copy commencing ..."
*      while not done and no errors do
*          begin
*              call [ioread] to get a buffer1 of data
*              if end-of-device status returned then
*                  begin
*                      set done flag to TRUE
*                      set status value to NULL_ERR
*                  end
*              if no errors then
*                  begin
*                      if first block read then
*                          begin
*                              determine actual size of data block
*                          end
*                      call [iowrit] to write buffer1 of data
*                      end
*                  end
*              call [dsperr] to print message "... copy complete"
*          return status value
*      end
*
*****/
```

2100-12-002

3.2 Get a String from the Input Line Window

```

/*****
*
*   Name:
*       inpstr
*
*   Function:
*       Obtains a string from the Input Line window of the monitor
*
*   Synopsis:
*       inpstr(string,prompt,length)
*
*   Input Parameters:
*       prompt:  prompt to be presented
*
*   Output Parameters:
*       string:  text string entered at the keyboard
*       *length: number of characters in string
*
*   Return Values:
*       NULL_ERR = okay
*
*   Functions Referenced:
*       clrinp:   clear input line window of display
*       printf:   print on IBM-AT monitor
*       waddstr:  add a string to a window
*       werase:   clear the window of all characters
*       winch:    get a character from the screen through a window
*       wgetch:   get a character from the keyboard through a window
*       wmove:    position cursor on window
*       wrefresh: update window on Monitor
*
*   Global Variables Referenced:
*       input_line:  Input Line window of display
*       pmt_txt:     text for prompt messages
*       prompt_line: Prompt Line window of display
*       prompt_table: pointers to text of prompt messages of Field Checking System
*
*   Description:
*       Prompts for search string, and reads in characters from keyboard until
*       the enter key is depressed, then returns former prompt message to the
*       display.
*
*****/
```


2100-12-002

```
* Algorithm:
*   begin
*       set return value to NULL_ERR
*       for each character of Prompt Line window do
*           begin
*               call [wmove] to move to next character
*               call [winch] to get character through window and save text
*           end
*       mark end of save text with null character
*       call [werase] to clear prompt line window
*       get prompt text from prompt table
*       call [waddstr] to add prompt text to window
*       call [wrefresh] to update prompt line on Monitor
*       call [clrinp] to clear input line window
*       while keyboard entry not carriage return do
*           begin
*               call [wgetch] to get keyboard character
*               if character is 'backspace' then
*                   begin
*                       if have not backed up to beginning of window then
*                           backup by decrementing text index pointer
*                       else
*                           call [printf] to signal bell -- cannot backup any further
*                       end
*                   end
*               else
*                   add character to text string
*               end
*           end
*       mark end of string with 'null' character
*       call [werase] to erase prompt line window
*       call [waddstr] to add former prompt text to prompt line window
*       call [wrefresh] to refresh to prompt line window
*       call [clrinp] to clear input line window
*       return status value
*   end
```

*****/

2100-12-002

3.3 Search a Data Block for a String

```

/*****
*
* Name:
*   sserch
*
* Function:
*   Searches a large text string for a match on a smaller text string
*
* Synopsis:
*   sserch(blk,str,b_end,s_count)
*
* Input Parameters:
*   blk:      large data block of characters to be searched
*   str:      text to be located in blk
*   b_end:    end of blk
*   s_count:  length of str
*
* Output Parameters:
*
* Return Values:
*   0 .. up = match found at position
*   GENERAL_ERR = no match found
*
* Functions Referenced:
*
* Global Variables Referenced:
*
* Description:
*   Searches block for a match of string and returns a status indicating the
*   character position in the block where the match starts
*
* Algorithm:
*   begin
*       set return value to NULL_ERR
*       if b_end - blk is less than s_count then
*           set return value to GENERAL_ERR
*       else
*           begin
*               set string pointer to 0
*               set block pointer to 0
*               set found flag to FALSE
*               while not found and block pointer is less than or equal to b_end do

```

2100-12-002

```
*      begin
*      if char at block pointer = char at string pointer then
*          begin
*              increment string pointer
*              if string pointer = s_count then
*                  set found flag to TRUE
*              else if block pointer + 1 is less than or equal to b_end and
*                  char at string pointer is not same as
*                  char at block pointer + 1 then
*                  set string pointer to 0
*              end
*          increment block pointer
*      end
*      if found then
*          set return value to block pointer - s_count
*      else
*          set return value to GENERAL_ERR
*      end
*      return status value
*  end
*
*****/
```

2100-12-002

3.4 Send Block of Data to Destination Device

```

/*****
*
* Name:
*   wrtblk
*
* Function:
*   Writes a block to specified device and handles any termination conditions
*
* Synopsis:
*   wrtblk(dst,buffer,bfr_len,done)
*
* Input Parameters:
*   dst:      LUN for destination device of data
*   buffer:    data to be written
*   bfr_len:  number of bytes to write
*   done:     flag indicating that last block to be written
*
* Output Parameters:
*
* Return Values:
*   NULL_ERR      = okay
*   TBL_FULL_ERR  = disk directory full
*   LIMIT_ERR     = attempt to write beyond bounds specified for hard disk
*   negative      = see [iowrit]
*
* Functions Referenced:
*   dsperr:      display a message on the Error Line window
*   ioctl:       perform low function on a device
*   iowrit:      write a buffer of data to a specified device
*   wgetch:      get a character through window
*
* Global Variables Referenced:
*   d_map0:      directory for rigid disk #0
*   d_map1:      directory for rigid disk #1
*   input_line:  Input line window of display
*   lun_tbl:     maps logical unit numbers to physical devices
*   rec_prm:     recording parameters and specifications
*   sdv_spc:     storage device specifications
*
* Description:
*   Writes a block of data to specified device and handles updating of
*   directory or changing of magnetic tape depending upon the storage

```

2100-12-002

```
*      medium.
*
* Algorithm:
*   begin
*     set return value to NULL_ERR
*     if first block read then
*       begin
*         if destination device is magnetic tape then
*           begin
*             calculate number of blocks which may be written to tape =
*               tape available / (blk size / recording density +
*                 interrecord gap)
*           end
*         else if device is one of the hard disks then
*           begin
*             determine corresponding directory for disk
*             find a free entry in directory
*             if no free entry then
*               call [dsperr] with "table full"
*             else
*               begin
*                 set logical address (lad) to last address written
*                   on disk
*                 if lad is less than lower bound or greater than upper
*                   bound of disk then
*                   call [dsperr] with "limit error"
*                 else
*                   begin
*                     set current disk pointer to lad
*                     call [lseek] to position physical heads on disk
*                     if not positioned at correct address then
*                       call [dsperr] with message "error at destination device"
*                     else
*                       begin
*                         set block record count to length of block
*                         calculate maximum number of blocks which may
*                           be written to the disk
*                         set line flag to -1
*                       end
*                     end
*                   end
*                 end
*             end
*           end
*         else
*           set number of blocks to maximum possible
*         end
*       end
*     if no errors then
```

2100-12-002

```
*      begin
*      increment block counter
*      if device is hard disk then
*      begin
*      if not done then
*      determine line number of block
*      if line number has changed or transfer is done or
*      max. number of blocks have been written then
*      begin
*      if line number is not -1 then
*      begin
*      save ending lad in directory
*      find next free entry
*      if no more free entries then
*      call [dsperr] with "table full"
*      end
*      if no errors and not done then
*      begin
*      save date in directory
*      save line number in directory
*      save block record length in directory
*      save starting lad of line in directory
*      set line flag to current line number
*      end
*      end
*      end
*      end
*      if no errors then
*      begin
*      if dest. device is magnetic tape and block counter has
*      surpassed maximum block number then
*      begin
*      call [ioctl] to place EOF marker on tape
*      call [dsperr] with message "change tape..."
*      set key to 0
*      set resume to FALSE
*      while not resume do
*      begin
*      call [wgetch] to get keyboard resume character
*      if key is 'f1' then
*      set resume flag to TRUE
*      end
*      call [dsperr] to clear error line
*      set block counter to 1
*      end
*      end
```

2100-12-002

```
*      if no errors and not done then
*          call [iowrit] to write buffer of data
*      if done
*          if output device is mag tape then
*              call [lioc1] to write end-of-file marker
*          else if output device is hard disk then
*              begin
*                  call [lseek] to position disk heads at directory
*                  call [iowrit] to rewrite updated directory
*              end
*          if done or error then
*              set block count to 0
*          return status value
*      end
*
*****/
```

3.5.1 Open a Device for Access

```

/*****
*
*   Name:
*       iopen
*
*   Function:
*       opens a specified device for input/output
*
*   Synopsis:
*       iopen(lun,mode,fld_ptr)
*
*   Input Parameters:
*       lun:      logical unit number of destination device
*       mode:     access mode of device (read, write, read/write)
*       fld_ptr:  field on screen containing device lun to be opened
*
*   Output Parameters:
*       d_map0:   Iomega hard disk #0 directory
*       d_map1:   Iomega hard disk #1 directory
*
*   Return Values:
*       NULL_ERR      = okay
*       GENERAL_ERR   = disk seek not performed correctly
*       NO_OPEN_ERR   = cannot open specified device
*       ( ***)        = see [ioread] for possible disk-related errors
*
*   Functions Referenced:
*       dsperr:  display error message on Error Line
*       ioclos:  close I/O device
*       ioctl:   issue command to a specified device
*       ioread:  read from a specified device
*       lseek:   position VENIX file/device pointer to a specified address
*       open:    low level unix routine to open a device
*
*   Global Variables Referenced:
*       buffer1: character buffer for device I/O
*       d_map0:  disk directory for Iomega drive #0
*       d_map1:  disk directory for Iomega drive #1
*       lun_tbl: table mapping logical unit numbers to physical devices
*       msg_prt: print screen messages to printer option flag
*
*   Description:

```


2100-12-002

* Opens the requested device: if the device is one of the two hard disks
* then the disk directory is read and the recording block size is determined
*

* NOTE: If device is the Star printer, is already opened, and the print-
* screen-messages-to-printer option is in effect, then no open will
* be performed and no error message will be issued (see [ioclos]).
*

* Algorithm:

```
* begin
*   set return value to NULL_ERR
*   if device is star printer, already open and print option in effect then
*     return status value
*   else if mode not write or read/write and device is not AT MONITOR then
*     begin
*       call [open] to open the desired device
*       if device not opened correctly then
*         begin
*           call [dsperr] with message "cannot open device"
*           clear lun table descriptor for device
*         end
*     else if device is hard disk #0 or hard disk #1 then
*       begin
*         set map pointer to corresponding device directory
*         set logical address pointer to 0
*         call [ioctl] to test unit
*         call [ioread] to read the disk directory
*         if error from [ioread] then
*           begin
*             call [ioclos] to close the device
*             call [dsperr] to erase error message on display
*           end
*       else
*         begin
*           set logical address pointer to starting lad of partition
*           call [lseek] to position VENIX addrs at correct lad on disk
*           if mode is "read" then
*             begin
*               set number of records/block to maximum possible
*               call [ioread] to read first data block in partition
*               if errors from [ioread] then
*                 call [ioclos] to close the device
*             else
*               begin
*                 find true length of a data block
*                 save true number of records/block in directory (bcnt)
*                 reset current lad pointer to starting lad of partition
```

2100-12-002

```
*               call [lseek] to reposition VENIX addr
*               end
*           end
*       end
*   end
*   return status value
* end
*
*****/
```

2100-12-002

3.5.2 Read from a Device

```

/*****
*
* Name:
*   ioread
*
* Function:
*   read a block of data from a specified device
*
* Synopsis:
*   ioread(lun,bfr,size)
*
* Input Parameters:
*   lun:    logical unit number of destination device
*   size:   number of bytes of data to be read
*
* Output Parameters:
*   bfr:    block of data into which to read
*
* Return Values:
*   NULL_ERR      = okay
*   GENERAL_ERR   = error encountered
*   END_SRC_DEV_STS = end of device encountered
*
* Functions Referenced:
*   dsperr:  display Error/Status message on Error Line of Monitor
*   read:    low level read from a device
*
* Global Variables Referenced:
*   d_map0:  hard disk drive #0 directory
*   d_map1:  hard disk drive #1 directory
*   lun_tbl: table mapping logical unit numbers to physical devices
*
* Description:
*   Obtains the input from the desired device, mapping the input from the
*   proper location on the device
*
* Algorithm:
*   begin
*       set return value to NULL_ERR
*       if device pointer is 0 or less then
*           call [dsperr] with message "error"
*       else if lun is mag tape #1 or mag tape #2 then

```

2100-12-002

```
*      begin
*      call [read] to read buffer
*      if return status from [read] is less than 0 then
*          call [dsperr] to display message "error at source device"
*      else if return status from [read] is 0 then
*          call [dsperr] with message "end of source device encountered"
*      else if status equals number of chars to be read then
*          reset status to NULL_ERR
*      end
*  else if LUN is hard disk #1 or hard disk #2 then
*      begin
*          set directory pointer to corresponding device map
*          if number of bytes to read is greater than or equal to maximum then
*              set number of bytes to read = block length + 255 divided by 256
*          else
*              round desired number of bytes off to multiple of 256
*          call [read] to get data
*          if status from [read] is negative then
*              call [dsperr] with "error at source device"
*          else if number of bytes read equal desired to be read then
*              if the read was part of opening the disk then
*                  reset status to NULL_ERR
*              else
*                  begin
*                      add number of records read to device counter
*                      if at end of device then
*                          call [dsperr] with "end of source device"
*                      else
*                          reset status to NULL_ERR
*                      end
*                  end
*              end
*          else
*              call [dsperr] with message "invalid device"
*          return status value
*      end
```

*****/

2100-12-002

3.5.3 Write to a Device

```

/*****
*
* Name:
*     iowrit
*
* Function:
*     writes a block of data to a specified device
*
* Synopsis:
*     iowrit(lun,bfr,size)
*
* Input Parameters:
*     lun:    logical unit number of destination device
*     bfr:    block of data to be written
*     size:   number of bytes of data to be written
*
* Output Parameters:
*
* Return Values:
*     NULL_ERR    = okay
*     GENERAL_ERR = device not open for access
*     INV_DEV_ERR = cannot write to device
*
* Functions Referenced:
*     dsperr:    display Error/Status message on Error Line of Monitor
*     wrefresh:  update Monitor to look like 'stdscr'
*     waddch:    add a character to the specified window
*     wmove:     move to a specific location within a window
*     write:     low level write to a device
*
* Global Variables Referenced:
*     d_map0:    Directory for hard disk #0
*     d_map1:    Directory for hard disk #1
*     lun_tbl:   Maps physical devices to logical unit numbers
*     present_area:  Presentation Area window of display
*
* Description:
*     Prepares the output for the desired device and maps the output to the
*     proper location on the device
*
* Algorithm:
*     begin

```

2100-12-002

```
*      set return value to NULL_ERR
*      if lun is null device then
*          return status
*      if file descriptor for device is zero and device is not the
*          IBM-AT monitor then
*          call [dsperr] with general error
*      else if lun is Start Printer or Versatec Printer or
*          Versatec Printer/Plotter then
*          begin
*              call [write] to print the partial buffer
*              if status is less than 0 then
*                  call [dsperr] with "end at destination device"
*              else if number of chars. written equals number requested then
*                  set status to NULL_ERR
*              end
*          else if lun is IBM-AT Monitor then
*              begin
*                  set index to 0
*                  set screen pointer to 0
*                  while index less than number of chars to be written do
*                      begin
*                          if character is carriage return or null then
*                              skip to next character
*                          else
*                              begin
*                                  call [wmove] to position cursor in presentation area
*                                  call [waddch] to add char to window
*                                  end
*                              if number of chars in screen causes a full screen then
*                                  begin
*                                      call [wrefresh] to update screen
*                                      set screen pointer to 0
*                                      end
*                              else
*                                  increment screen pointer
*                              end
*                          for unfilled in portion of screen do
*                              begin
*                                  call [wmove] to move to next unfilled location
*                                  call [waddch] to add a space to the window
*                                  end
*                              call [wrefresh] to update screen
*                              end
*          else if lun is mag tape #0 or mag tape #1 then
*              begin
*                  call [write] to write buffer
```

2100-12-002

```
*      if return status is less than zero then
*          call [dsperr] to display message "error at destination device"
*      else if return status same as # of bytes desired to be written then
*          set return status to NULL_ERR
*      end
*  else if lun is hard disk #0 or hard disk #1 then
*      begin
*          determine respective disk directory involved in operation
*          round up number of character to write to nearest 256 increment
*          call [write] to write buffer
*          if return status is less than zero then
*              call [dsperr] to display message "error at destination device"
*          else if return status same as # of bytes desired to be written then
*              begin
*                  increment file pointer based on number of records written
*                  if at end of disk then
*                      call [dsperr] with "error at destination device"
*                  else
*                      set return status to NULL_ERR
*                  end
*              end
*          end
*      else
*          call [dsperr] with message "invalid device"
*      return status value
*  end
*
*****/
```

3.5.4 Close a Device from Access

```

/*****
*
*   Name:
*       ioclos
*
*   Function:
*       closes a specified device for input/output
*
*   Synopsis:
*       ioclos(lun)
*
*   Input Parameters:
*       lun:   logical unit number of destination device
*
*   Output Parameters:
*
*   Return Values:
*       NULL_ERR = okay
*
*   Functions Referenced:
*       close:   low level unix routine to close a device
*
*   Global Variables Referenced:
*       lun_tbl: table mapping logical unit numbers to physical devices
*       msg_prt: error console message printing flag
*
*   Description:
*       Closes the requested device.
*
*   Algorithm:
*       begin
*           set return value to NULL_ERR
*           if device descriptor is non-zero and msg_prt is not set and
*               device is not the printer then
*               call [close] to close the desired device
*           return status value
*       end
*
*   NOTE: When the hard copy screen message printing option is in effect,
*         the Star printer will remain opened until the Field Checking System
*         is concluded. Any calls to [ioclos] will ignore the printer.
*
*****/

```


2100-12-002

*****/

2100-12-002

4.0 BLOCK PRINTING ROUTINES

4.1 Select Print Format

```

/*****
*
*   Name:
*       fmtspc
*
*   Function:
*       gets printing format
*
*   Synopsis:
*       fmtspc(fileid)
*
*   Input Parameters:
*       fileid:   coded portion of file name for required display file
*
*   Output Parameters:
*
*   Return Values:
*       NULL_ERR = okay
*
*   Functions Referenced:
*       dsperr:   display error text on screen
*       dspscr:   places menu/display and prompt on screen
*       fqyspc:   dump data blocks at a specified frequency
*       wgetch:   get a character through a window
*
*   Global Variables Referenced:
*       air_chk_set:  array of valid airborne data block chars and print header
*       diu_chk_set:  array of valid diurnal data block chars and print header
*       input_line:   input line window of display
*
*   Description:
*       Presents Block Printing Menu and handles Operator choices until
*       CTRL Z entered
*
*   Algorithm:
*       begin
*           set return value to NULL_ERR
*           set input response to 0
*           while input response is not CTRL Z or CTRL C do

```

2100-12-002

```
*      begin
*      call [dspscr] to display menu
*      if error from [dsperr] then
*          return status value
*      call [wgetch] to get Operator choice
*      case Operator choice of
*          1 : call [fqyspc] with airborne tables
*          2 : call [fqyspc] with diurnal tables
*          3 : call [fqyspc] with dummy tables
*          CTRL C:
*          CTRL Z: no operation
*          default: call [dsperr] to display message "option not available"
*      end
*      return status value
*  end
*
```

*****/

2100-12-002

4.2 Define Frequency

```

/*****
*
*   Name:
*       fqyspc
*
*   Function:
*       Define print frequency for block printing
*
*   Synopsis:
*       fqyspc(fileid,ftype,char_set)
*
*   Input Parameters:
*       fileid:    coded portion of file name for display file
*       ftype:     format type to be printed (ie. airborne, diurnal, unformatted)
*       char_set:  character set against which to validate data
*
*   Output Parameters:
*
*   Return Values:
*       NULL_ERR = okay
*
*   Functions Referenced:
*       chgval:    change a screen value in the Presentation Area
*       dmpblk:    print the blocks of data from the given source device
*       dsperr:    display error message on Error Line
*       dspscr:    present menu on Monitor
*       echo:      enable keyboard echo
*       inpstr:    get a string from the keyboard
*       movcsr:    move the visible cursor on the Monitor
*       noecho:    disable keyboard echo
*       txtdec:    convert text to decimal integer
*       wgetch:    get a character from the keyboard through a window
*       wvalue:    write values displayed on monitor to file
*
*   Global Variables Referenced:
*       input_line: Input Line window of display
*       lun_tbl:     Table mapping logical unit numbers to physical devices
*       scr:         IBM-AT Monitor window text
*
*   Description:
*       Presents the Block Printing Frequency Menu, obtains the Operator's
*       choice and calls the appropriate routine to dump the block of data

```

```

*
* Algorithm:
*   begin
*     set return value to NULL_ERR
*     call [dspscr] to display menu
*     if error from [dspscr] then
*       return status value
*     set input response to 0
*     while input response is not ((CTRL Z and no errors) or CTRL C) do
*       begin
*         call [wgetch] to get keyboard input
*         call [movcsr] to move cursor if required
*         if return status from [movcsr] is NOT_CRSR_STS or input is CTRL Z then
*           begin
*             call [chgval] to make change on display
*             if return from [chgval] is INP_LINE_STS or input is CTRL Z then
*               begin
*                 get error checking response from screen
*                 if character is affirmative then
*                   set error checking flag to TRUE
*                 else if character is negative then
*                   set error checking flag to FALSE
*                 else
*                   call [dsperr] with message "invalid response"
*                 if error checking flag was set properly then
*                   begin
*                     get source LUN from screen
*                     get destination LUN from screen
*                     if source LUN not in range then
*                       call [dsperr] with "invalid character"
*                     else if destination LUN not in range then
*                       call [dsperr] with "invalid character"
*                     else if source LUN not valid device then
*                       call [dsperr] with "invalid device"
*                     else if destination LUN not valid device then
*                       call [dsperr] with "invalid device"
*                     else
*                       begin
*                         call [dsperr] to clear error line
*                         case key of
*                           '1' : set frequency = 1;
*                           '2' : set frequency = 10;
*                           '3' : set frequency = 100;
*                           '4' : begin
*                             call [echo] to enable keyboard echoing
*                             call [inpstr] to get number from keyboard

```

2100-12-002

```
*          call [noecho] to disable keyboard echoing
*          call [txtdec] to convert string
*          if no errors from [txtdec] then
*              set frequency = return parameter (integer)
*          end
*      '5' : if error checking flag is FALSE then
*          call [dsperr] with "ambiguous specification"
*              (error checking not requested)
*      else
*          set frequency = 0; (indicates bad blocks)
*      default: call [dsperr] with "option not available"
*  if no errors detected then
*      begin
*          call [dmpblk] with frequency, char_set, and format type
*          reset status to NULL_ERR
*      end
*  end
*  end
*  end
*  end
*  if input is CTRL Z then
*      call [wvalue] to write changed values back to value file
*  return status value
*  end
*
*****/
```

4.3 Dumping Process

```

/*****
*
* Name:
*     dmpblk
*
* Function:
*     To print a block of data on the output device
*
* Synopsis:
*     dmpblk(src,dst,frequency,ftype,d_set,err_chk)
*
* Input Parameters:
*     src:          LUN of source device
*     dst:          LUN of destination device
*     frequency:    frequency at which blocks are to be dumped
*     ftype:        format type to be printed
*     d_set:        character set against which to validate data & print header
*     err_chk:      error checking of all blocks requested
*
* Output Parameters:
*
* Return Values:
*     NULL_ERR      = okay
*     SRC_DEV_ERR   = error at source device
*     DST_DEV_ERR   = error at destination device
*     negative      = error encountered
*
* Functions Referenced:
*     asctime:      convert time to ascii
*     chkblk:       check data block for errors
*     ioclos:       close an i/o device
*     dsperr:       display a message on the error line window
*     hextxt:       convert hexadecimal integer to text string
*     ioopen:       open an i/o device
*     ioread:       read a block of data from an i/o device
*     iowrit:       write a block of data to an i/o device
*     sprintf:      formatted print to string
*     sserch:       find a match on a string in a piece of text
*     strlen:       determine length of string
*     time:         get system time
*
* Global Variables Referenced:

```

2100-12-002

```
*      buffer1:      data block storage area
*      buffer2:      data block storage area
*      d_map0:       hard disk directory 0
*      d_map1:       hard disk directory 1
*      err_txt:      contains actual text for all FCS error messages
*      error_table:  contains pointers to text for all FCS error messages
*      lun_tbl:      table mapping logical unit numbers to physical devices
*      rec_prm:      recording parameter table of codes and names
*
```

* Description:

```
*      Opens I/O devices, reads and verifies data blocks, and prints
*      data in desired format until end of source device
*      or error is encountered
*
```

* Algorithm:

```
*      begin
*          set return value to NULL_ERR
*          call [ioopen] to open the source device
*          if no error from [ioopen] then
*              begin
*                  call [ioopen] to open the destination device
*                  if error from [ioopen] then
*                      call [ioclos] to close the source device
*                  end
*              if no errors and destination device not Standard Message Console then
*                  begin
*                      search lun table for Message Console LUN definition
*                      if not found then
*                          call [dsperr] with "error"
*                      else
*                          call [ioopen] to open device
*                      if errors then
*                          begin
*                              call [ioclos] to close source device
*                              call [ioclos] to close destination device
*                          end
*                      end
*                  if errors then
*                      return status value
*                  call [dsperr] with message "block printing commencing ..."
*                  set block counter to 0
*                  set line counter to 0
*                  set quit flag to FALSE
*                  set block length to maximum length
*                  while not quit do
*                      begin
```



```

*      set print flag to 0
*      call [ioread] to get data from source device
*      if return status from [ioread] is error then
*          set quit flag to TRUE
*      else
*          begin
*              if status from [ioread] is end-of-device then
*                  begin
*                      set print flag to 1
*                      call [dsperr] with "... block printing complete"
*                      set quit flag to TRUE
*                  end
*              if first block to be processed then
*                  begin
*                      set up pointer to text for dump type
*                      call [sprintf] to copy text to block id buffer
*                      call [strlen] to determine length of block id
*                      call [strlen] to determine length of data block
*                          (end marked by null character)
*                      if length of block less header DIV 10 is not whole then
*                          set print flag to invalid block length
*                      else if format type is airborne or diurnal then
*                          begin
*                              calculate unformatted record length (url)
*                              set formatted data buffer to spaces
*                              set temporary buffer (tmp buf) for header parameter
*                                  codes to spaces
*                              clear header length and mnemonic pointer
*                              while elements of header to be processed do
*                                  begin
*                                      add number of preceding spaces to header length
*                                      set formatted record pointer (frp) for mnemonic to
*                                          current length of header
*                                      if not first parameter then
*                                          begin
*                                              set "full" to the length of current parameter value
*                                              set "available" to the number of character positions
*                                                  between the start of the current parameter and
*                                                  the end of the previous parameter less 1 (to be
*                                                  left as a space dividing the columns)
*                                              set number of digits to print for the parameter as
*                                                  the lesser of full and available
*                                              set the formatted record length (frl) to the
*                                                  current header length plus the number of digits
*                                                  to be printed for the previous parameter
*                                          end
*                                  end
*                  end

```

```

*           set temporary pointer to current header length
*           if current parm. being processed is not NULL then
*               begin
*                   copy parm. code to tmp buf
*                   copy the mnemonic to the header
*                   add length of mnemonic to header length
*                   call [sserch] to find the parameter in data block
*                   if not found then
*                       begin
*                           set print flag to missing parameter
*                           set quit flag to TRUE
*                       end
*                   else
*                       save (for later use) the pointer returned from
*                           [sserch] as unformatted record pointer (urp)
*                   end
*               get next header element
*           end
*           add carriage return at header length position
*           for number of characters in header do
*               begin
*                   add recording parameter code to header at position
*                   add underline character to header at correct position
*               end
*           place carriage return characters in header after line
*               containing codes and line containing underline char.
*           re-adjust header length for two new lines added
*           add one more to frl to account for carriage return
*           for each of ten records
*               at the end of each frl position add a carriage return
*           add a final carriage return at end of formatted block
*           determine number of lines making up header
*       end
*   end
*   if print flag is not negative and format type is formatted and
*       error checking is requested then
*       begin
*           call [chkblk] to check for bad characters, parameters, etc.
*           if error from [chkblk] then
*               set print flag to error status
*           end
*       if print flag is not set and frequency is not every bad block and
*           time to print and format type is not unformatted then
*           begin
*               set print flag to 1
*               set header parameter pointer to zero

```

2100-12-002

```
*      while not all parms of header have been included and
*      print flag not negative do
*      begin
*      if parameter not NULL then
*      begin
*      while counter is less than 10
*      and print flag not negative do
*      copy data (with correct spacing) to frmt bfr
*      end
*      increment header parameter pointer
*      end
*      end
*      if print flag set and frequency is zero or time to print then
*      begin
*      copy date (day) to block id
*      copy date (year) to block id
*      if format type is airborne then
*      copy line number to block id
*      end
*      if print flag is 1 and frequency is 0 then
*      set print flag to 0 (not a bad block - do not dump)
*      else if frequency not zero and time to print and format
*      is unformatted then
*      set print flag to 1
*      else if print flag is negative and frequency is not every bad
*      and not time to print a block then
*      begin
*      call [time] to get the system time and date
*      call [asctime] to convert to ascii
*      copy time to print buffer
*      copy Bad Block message to print buffer
*      call [dectxt] to add block number in error to print buffer
*      mark end of print buffer with carriage return
*      call [iowrit] to write buffer to Message Console
*      leave spaces at beginning of next line of message
*      copy corresponding error message to print buffer
*      mark end of print buffer with carriage return
*      call [iowrit] to write buffer to Message Console
*      copy first of header message to output buffer
*      copy header to output buffer
*      copy end of header message to output buffer
*      if source device is a hard disk then
*      begin
*      set up pointer to disk directory
*      copy logical address message to output buffer
*      call [hexxtxt] to add disk LAD to output buffer
```

2100-12-002

```
*           end
*           add carriage return to end of output buffer
*           call [iowrit] to send output buffer to destination device
*           set print flag to 0 (no dump)
*           end
*       if print flag not zero then
*       begin
*           call [dectxt] to add block number to block id string
*           call [dectxt] to add length to block id string
*           determine number of lines to be printed on page
*           if number of lines printed will overflow page then
*           begin
*               call [iowrit] to print a form feed
*               clear line counter
*           end
*           add number of lines to be printed to line counter
*           call [iowrit] to print block id
*           add two carriage returns to end of unformatted block
*           if print flag is positive then
*               if format type is unformatted then
*                   call [iowrit] to dump unformatted block
*               else
*                   call [iowrit] to print formatted data block
*           else
*               begin
*                   copy required error message text to a temporary buffer
*                   if source device is a hard disk then
*                   begin
*                       set up pointer to disk directory
*                       copy logical address message to output buffer
*                       call [hextxt] to add disk LAD to output buffer
*                   end
*                   add carriage return to end of output buffer
*                   call [iowrit] to print error message
*                   call [iowrit] to print unformatted data block
*               end
*           end
*       end
*       increment block counter
*       end
*   end
*   call [ioclos] to close the source device
*   call [ioclos] to close the destination device
*   if destination device not Standard Message Console then
*       call [ioclos] to close device
*   return status value
* end
```

2100-12-002

*
*****/

4.3.1 Check a Data Block for Errors

```

/*****
*
* Name:
*   chkblk
*
* Function:
*   To check a block of data for correct characters and parameters
*
* Synopsis:
*   chkblk(buffer,first,chk_set,psn,nbr)
*
* Input Parameters:
*   buffer:      data block to be checked
*   first;       first block to be checked flag
*   chk_set:     checking set against which to validate data parameters
*
* Output Parameters:
*   psn:         character position in data block of error
*   nbr:         number of characters in error (or length of field in error)
*
* Return Values:
*   NULL_ERR      = okay
*   INV_CHAR_ERR  = invalid character or parameter value
*   INV_P_CODE_ERR = invalid parameter code
*   INV_BLK_LEN_ERR = invalid block length
*   NO_P_CODE_ERR  = missing parameter code
*
* Functions Referenced:
*   dsperr:      display a message on the error line window
*   sserch:      find a match on a string in a piece of text
*   txtdec:      convert text to decimal integer
*   txthex:      convert text to hexadecimal integer
*   vdatbk:      validate the data block for correct characters
*
* Global Variables Referenced:
*   rec_prm:      recording parameter table of codes and names
*
* Description:
*   Checks the data block for invalid characters and then checks each
*   parameter for correct format and digit type
*
* Algorithm:

```

```

*      begin
*      set return value to NULL_ERR
*      set error position to 0
*      set error character count to 0
*      if first block to be processed then
*      begin
*      calculate length of data block ( end marked by null character or
*      till character (~) )
*      if length of block less header DIV 10 is not whole then
*      call [dsperr] with message "invalid block length"
*      else
*      begin
*      calculate unformatted record length (url)
*      while elements of header to be processed and no errors do
*      if element is not the NULL parameter then
*      begin
*      call [sserch] to find the parameter in data block
*      if not found then
*      call [dsperr] with message "missing parameter code"
*      end
*      end
*      end
*      if no errors then
*      begin
*      call [vdatbk] to verify all characters in the data block
*      if return status from [vdatbk] not zero then
*      begin
*      set error position to status minus 1 (buffer position)
*      set number of characters in error to 1
*      call [dsperr] with message "invalid character"
*      end
*      else
*      begin
*      while not done processing all parameters and no errors do
*      if header parameter not NULL parameter then
*      begin
*      while not all ten records of recording block have been
*      examined and no errors do
*      begin
*      if parameter code is not valid then
*      begin
*      set position to erroneous parameter code
*      set number of erroneous characters to 2
*      call [dsperr] with "invalid parameter code"
*      end
*      else if an error is detected during conversion from

```

2100-12-002

```
*          text to integer (call [txthex] or [txtdec]) then
*          begin
*          set position to erroneous parameter value
*          set number of error characters to length of parm.
*          end
*          end
*          end
*          end
*          return status value
*          end
*
***/
```


2100-12-002

4.3.1.1 Check Against Valid Character Set

```

/*****
*
* Name:
*   vdatbk
*
* Function:
*   Verify characters in data block
*
* Synopsis:
*   vdatbk(dat_blk, val_chars, blk_size)
*
* Input Parameters:
*   dat_blk:   pointer to a block of characters to be verified
*   val_chars: valid characters which may be contained in data block
*   blk_size:  length of data block
*
* Output Parameters:
*
* Return Values:
*   NULL_ERR    = okay
*   positive    = character position of error ( 1 .. n )
*
* Functions Referenced:
*
* Global Variables Referenced:
*
* Description:
*   Verifies that all characters in the data block are valid as defined
*   by the character checking set
*
* Algorithm:
*   begin
*       set return value to NULL_ERR
*       set data-block-character pointer (blk_ptr) to 0
*       while no errors and blk_ptr is less than block size do
*           if character in data block has no flag set for it in the character
*               checking set then
*               set status to "invalid character"
*           else
*               increment blk_ptr
*       return status value
*   end
*
*****/
```

2100-12-002

*
*****/

2100-12-002

5.0 PLOTTING ROUTINES

5.1 Present Plotting Menu

```

/*****
*
*   Name:
*       pltmnu
*
*   Function:
*       Defines plot specifications and performs plot
*
*   Synopsis:
*       pltmnu(fileid)
*
*   Input Parameters:
*       fileid:  coded portion of file name for display file
*
*   Output Parameters:
*
*   Return Values:
*       NULL_ERR = okay
*       negative = error encountered
*
*   Functions Referenced:
*       dsperr:   clear error line of display
*       dspscr:   present menu/display on the Monitor
*       pltspc:   define plot specifications
*       wgetch:   get a character from the keyboard through the window
*
*   Global Variables Referenced:
*       input_line:  Input Line window of display
*
*   Description:
*       Presents the Plot Menu and presents the Operator choice of Plot
*       Specifications display
*
*   Algorithm:
*       begin
*           set return value to NULL_ERR
*           set input response to 0
*           while input response is not CTRL Z or CTRL C do
*               begin

```

2100-12-002

```
*      call [dspscr] to display menu
*      if error from [dspscr] then
*          return status value
*      call [wgetch] to get keyboard input
*      case input of
*          '1' : call [pltspc] to present desired display
*          '2' : call [pltspc] to present desired display
*          '3' : call [pltspc] to present desired display
*          CTRL C:
*          CTRL Z:
*              no operation
*          default: call [dsperr] with "invalid option"
*      end
*      return status value
*  end
*
*****/
```

5.2 Define Plot Specifications

```

/*****
*
*   Name:
*       pltspc
*
*   Function:
*       Defines plot specifications and performs plot
*
*   Synopsis:
*       pltspc(fileid)
*
*   Input Parameters:
*       fileid:  coded portion of file name for display file
*
*   Output Parameters:
*
*   Return Values:
*       NULL_ERR = okay
*       negative = see [dspscr]
*
*   Functions Referenced:
*       chgval:  change a screen value in the Presentation Area
*       dmpplt:  perform plotting process
*       dspscr:  present menu/display on the Monitor
*       echo:    enable keyboard input echoing
*       movcsr:  move the visible cursor on the Monitor
*       noecho:  disable keyboard input echoing
*       pltchk:  verify the plot specifications defined by the Operator
*       wgetch:  get a character from the keyboard through the window
*       wvalue:  write changed values of Presentation Area to value file
*
*   Global Variables Referenced:
*       input_line:  Input Line window of display
*
*   Description:
*       Presents the Plot Specifications Menu and accepts and validates Operator
*       input until a key is depressed in the Input Line at which time the plotting
*       process will commence
*
*   Algorithm:
*       begin
*           set return value to NULL_ERR

```

2100-12-002

```
*      call [dspscr] to display menu
*      if error from [dspscr] then
*          return status value
*      set input response to 0
*      call [noecho] to turn off keyboard echoing
*      while input response is not ((CTRL Z & no errors) or CTRL C) do
*          begin
*              call [wgetch] to get keyboard input
*              call [movcsr] to move cursor if requested
*              if status from [movcsr] is NOT_CRSR_STS or input is CTRL Z then
*                  begin
*                      call [chgval] to modify screen
*                      if status from [chgval] is INF_LINE_STS or input is CTRL Z then
*                          begin
*                              call [pltchk] to verify the plot specifications
*                              if no error from [pltchk] and input not CTRL Z and input
*                                  is 'P' then
*                                  call [dmpplt] to perform plot process
*                              end
*                          end
*                      end
*                  end
*              if input response is CTRL Z then
*                  call [wvalue] to save specifications
*                  call [echo] to turn on keyboard echoing
*                  return status value
*              end
*          end
*      *****/
```

2100-12-002

5.2.1 Verify Plot Specifications

```

/*****
*
* Name:
*   pltchk
*
* Function:
*   Verifies plot specifications
*
* Synopsis:
*   pltchk(plt_tbl)
*
* Input Parameters:
*
* Output Parameters:
*   plt_tbl:  table containing parms, scales, and positions for plotting
*
* Return Values:
*   NULL_ERR      = okay
*   INV_CHAR_ERR  = invalid character defining logical unit
*   INV_P_CODE_ERR = undefined parameter code specified
*   INV_PLOT_POSTN_ERR = plot position not within defined limits
*   INV_RESPONSE_ERR = invalid response
*   LIMIT_ERR     = vertical scale not within defined limits
*   NO_PARM_ERR   = missing parameter code
*
* Functions Referenced:
*   dsperr:  display message on Error/Status Line of Monitor
*   prmtch:  find match on parameter code in parameter table
*   txtdec:  convert text string to decimal integer value
*
* Global Variables Referenced:
*   rec_parm: recording parameters, lengths, types etc.
*   scr:      attributes and descriptions of fields on screen
*
* Description:
*   Verifies that the plot specifications set up by the Operator via the
*   keyboard and screen are valid
*
* Algorithm:
*   begin
*       set return value to NULL_ERR
*       set plot table index (pti) to 0

```

```

*      set field layout table index (flti) to the top, left corner
*      while no errors and length of field not 1 do
*          begin
*              call [prmtch] to verify the parameter code
*              if error returned from [prmtch] then
*                  call [dsperr] to display message "invalid parameter code"
*              else
*                  begin
*                      save rec. parm. table index in plot table
*                      call [txtdec] to convert plot scale to decimal integer
*                      if no error from [txtdec] then
*                          begin
*                              if first parameter then
*                                  begin
*                                      if scale is zero then
*                                          begin
*                                              if not null parameter then
*                                                  set successive difference flag in plot table
*                                              else
*                                                  call [dsperr] with "missing parameter"
*                                          end
*                                      else
*                                          clear successive difference flag in plot table
*                                  end
*                              save scale in plot table
*                              move to next field at right
*                              end
*                          if no errors and parameter is not the null parameter then
*                              begin
*                                  call [txtdec] to convert position to decimal integer
*                                  if no error from [txtdec] then
*                                      begin
*                                          if value greater than MAX_SCT or less than 0 then
*                                              call [dsperr] with message "invalid plot position"
*                                          else
*                                              begin
*                                                  save position as start in plot table
*                                                  call [txtdec] to convert position to decimal integer
*                                                  if no error from [txtdec] then
*                                                      begin
*                                                          if value greater than MAX_SCT or less than 0 or
*                                                             value less than start position then
*                                                              call [dsperr] with message "invalid plot position"
*                                                          else
*                                                              save stop sector position in plot table
*                                                      end
*                                                  end
*                                              end
*                                          end
*                                      end
*                                  end
*                              end
*                          end
*                      end
*                  end
*              end
*          end

```


2100-12-002

```
*           end
*         end
*       end
*     increment pti
*     move to next field at right
*   end
* end
* if no errors then
*   begin
*     if no errors then
*       begin
*         { vertical scale lines }
*         if text is 'y' or 'Y' then
*           set scale lines flag to TRUE in plot_table
*         else if text is 'n' or 'N' then
*           set scale lines flag to FALSE in plot_table
*         else
*           call [dsperr] to display message "invalid response"
*         end
*       if no errors then
*         begin
*           move field pointer to next at right
*           { vertical scale parameter }
*           call [txtdec] to convert to decimal value
*           if no error from conversion then
*             if converted value is greater than MAX_SCL then
*               call [dsperr] with message "limit error"
*             else
*               save converted value in plot table
*             end
*           if no errors then
*             begin
*               move field pointer to next at right
*               { fiducial parameter }
*               call [prmtch] to verify code
*               if return code from [prmtch] is negative then
*                 call [dsperr] to display message "invalid parameter code"
*               else
*                 begin
*                   save index into rec parm table as fiducial parm in plot table
*                   { fiducial parameter scale }
*                   call [txtdec] to convert scale to decimal value
*                   if no error from [txtdec] then
*                     save scale in plot table
*                   end
*                 end
*               end
*             end
*           end
*         end
*       end
*     end
```

2100-12-002

```
*      if no errors then
*      begin
*      set field pointer to next at right
*      { source lun }
*      get specified lun character from screen
*      if lun is smaller than smallest valid lun or
*      larger than largest valid lun then
*      call [dsperr] to display message "invalid character"
*      else
*      save source lun in plot table
*      end
*      end
*      return status value
*      end
*
```

*****/

5.2.2 Verify Parameter Code Against Recording Parameter Table

```

/*****
*
* Name:
*   prmtch
*
* Function:
*   Match a parameter code against recording parameter table
*
* Synopsis:
*   prmtch(cde,tbl_idx)
*
* Input Parameters:
*   cde:   recording parameter code to be matched in rec. parm. table
*
* Output Parameters:
*   tbl_idx:  index in recording parameter table of string matched
*
* Return Values:
*   NULL_ERR   = okay
*   GENERAL_ERR = no match
*
* Functions Referenced:
*
* Global Variables Referenced:
*   rec_parm:  table of recording parameter names and codes
*
* Description:
*   Attempts to match a recording parameter code against the entries
*   in the recording parameter table.  If a match is found the table
*   index is returned, otherwise the index returned is -1
*
* Algorithm:
*   begin
*       set return value to NULL_ERR
*       set search variable to NULL_PTR
*       set recording parameter table index (rpti) to 0
*       while search variable remains NULL_PTR and
*           not at end of recording parameter table do
*           if input string is same as parameter code at rpti then
*               set search variable to rpti
*           else
*               increment rpti

```

2100-12-002

```
*      if no match found then
*          set return status to GENERAL_ERR
*      set return parameter to search variable
*      return status value
*  end
*
*****/
```

5.3 Plotting Process

```

/*****
*
* Name:
*     dmpplt
*
* Function:
*     Dump plot to output device
*
* Synopsis:
*     dmpplt(plt_tbl)
*
* Input Parameters:
*     plt_tbl:  contains all data required to create plot
*
* Output Parameters:
*
* Return Values:
*     NULL_ERR      = okay
*     GENERAL_ERR   = cannot find versatec printer/plotter in lun table
*     INV_DEV_ERR   = source device not one of valid input devices
*     INV_P_CODE_ERR = invalid parameter code specified
*     NO_P_CODE_ERR = missing parameter code in data block
*     negative      = see [ioopen]
*
* Functions Referenced:
*     dsperr:  display error message on the monitor
*     ioclos:  close an I/O device
*     ioopen:  open a low level device
*     ipower:  integer exponentiation
*     ioread:  read data from a requested device
*     iowrit:  write data to a device
*     sendln:  send an entire line of spp data to device
*     sserch:  locate a string in a data block
*     txtdec:  convert text string to decimal value
*     txthex:  convert text string to hex value
*
* Global Variables Referenced:
*     buffer1:  data block storage area
*     buffer2:  data block storage area (used to store plot header text)
*     lun_tbl:  table mapping logical unit numbers to physical devices
*     rec_prms: recording parameters and specifications
*
*****/

```

2100-12-002

```
* Description:
*   Read data from the source device, extracts plot data, and formats and
*   dumps this data to the Versatec plotter.
*
* Algorithm:
*   begin
*     set return value to NULL_ERR
*     if source device not one of magnetic tape units or hard disks then
*       call [dsperr] with "invalid device"
*     else
*       begin
*         call [ioopen] to open source device
*         if no error from [ioopen] then
*           begin
*             set plotter pointer to NULL_PTR
*             set printer pointer to NULL_PTR
*             while (versatec plotter pointer not changed or
*                   versatec printer pointer not changed) and
*                   not at end of lun table do
*               begin
*                 if lun table entry is the versatec printer/plotter then
*                   set plotter pointer to table entry
*                 else if lun table entry is versatec printer then
*                   set printer pointer to table entry
*                 move to next entry in table
*               end
*             if not both pointers found then
*               call [dsperr] with message "general error"
*             else
*               begin
*                 call [ioopen] to open destination device
*                 if error from [ioopen] then
*                   call [ioclos] to close the source device
*                 end
*               end
*             end
*           end
*         if errors then
*           return status
*         set buffer2 to be used as plot header text to all spaces
*         set second to last character of each text line in buffer to line feed
*         set last character of each text line in buffer2 to NULL
*         for each position in print line do
*           if position is a unit of ten then
*             if position is a unit of 100 then
*               scale and save the 'tens' char in a scale line hdr (buffer2)
*             else
```

2100-12-002

```
*           save the 'tens' character in a scale line hdr (buffer2)
* for each sector of plot line do
*   begin
*     clear all bits in plot line buffer
*     clear all bits in a null plot line buffer
*   end
* clear plot memory array
* set up plot header #1 text (in buffer2)
* if plot type is successive difference then
*   begin
*     copy successive difference text to header #1
*     clear successive difference variable table
*     set successive difference number pointer to 0
*   end
* set up plot header #2 text (in buffer2)
* call [dectxt] to copy vertical scale to header #2 (in buffer2)
* call [dectxt] to copy fiducial interval to header #2 (in buffer2)
* copy fiducial parameter units to header #2 (in buffer2)
* set flight line flag to negative value
* set block length to maximum value
* set plot line counter to 0
* set plot line counter to 0
* set record counter to 0
* set done flag to FALSE
* set fiducial marker print flag to FALSE
* call [dsperr] with message "plotting commencing..."
* while no errors and not done do
*   begin
*     if record counter is 0 (ie. no records left to process) then
*       begin
*         call [ioread] to get a data block
*         if error or end of source device status from [ioread] then
*           set done flag to TRUE
*         else if first block then
*           begin
*             calculate true length of data block
*             set plot parameter pointer to beginning of table
*             while not at end of plot parameter table and no errors do
*               if a parameter is defined for table entry then
*                 begin
*                   if vertical scale lines desired then
*                     begin
*                       set bit for start of sector position in plot line bfr
*                       set bit for start of sector position in plot line bfr
*                     end
*                   if first parameter or plot type is not succ. diff. then
```

```

*           begin
*           call [sserch] find parameter code in data block
*           if no match found from [sserch] then
*               call [dsperr] with "missing parameter code"
*           else
*               save index into data block in plot table
*           end
*       if not first parameter or not success. diff. plot then
*       begin
*       copy mnemonic to mnemonic print line text (buffer2)
*       if parameter type is hexadecimal then
*           call [hextxt] to copy plot parameter scale to
*           scale line text (buffer2)
*       else
*           call [dectxt] to copy plot parameter scale to
*           scale line text (buffer2)
*       copy plot parameter units to unit line text (buffer2)
*       end
*   end
* if no errors then
*   begin
*   call [sserch] to find fiducial parameter in data block
*   if not found from [sserch] then
*       call [dsperr] with message "missing parameter code"
*   else
*       begin
*       call [sserch] to find second occurrence of fiducial parm
*       subtract the two positions to calculate record length
*       save index into data block for fid. parm. in plot table
*       call [sserch] to find line number in data block
*       call [sserch] to find date in data block
*       call [sserch] to find heading in data block
*       end
*   end
* end
* if no errors then
*   begin
*   if line number was found in data block then
*       begin
*       call [txthex] to convert line number to value
*       if no errors and line number not same as old number then
*           begin
*           set old line number to new value
*           clear the print line text (buffer2)
*           place carriage return and null at end of text (buffer2)
*           call [ioclos] to close plotter

```



```

*      call [iioopen] to open printer
*      call [iowrit] to write a form feed to printer
*      add line number to plot header #1 text (buffer2)
*      if heading parameter found in data block then
*          begin
*              call [txtdec] to get heading value
*              if no errors from [txtdec] then
*                  begin
*                      calculate position of heading code in print line
*                      clear heading code position in print buffer
*                      if heading is greater than 292.5 degrees and
*                          less than 360 degrees or
*                          greater/equal 0 degrees and
*                          less than 67.5 then
*                          set code to North and increment print position
*                      else if heading is greater than 112.5 degrees and
*                          less than 247.5 degrees then
*                          set code to South and increment print position
*                      if heading is greater than 22.5 degrees and
*                          less than 157.5 degrees then
*                          set next code to East
*                      else if heading is greater than 202.5 degrees and
*                          less than 337.5 degrees then
*                          set next code to West
*                  end
*              end
*          end
*      if date exists in data block then
*          add date to plot header #1 text (buffer2)
*      call [iowrit] to output header #1
*      call [iowrit] to output blank line
*      call [iowrit] to output header #2
*      call [iowrit] to output blank line
*      call [iowrit] to output blank line
*      call [ioclos] to close the printer
*      call [iioopen] to reopen the plotter
*      for each sector of plot line do
*          mark the beginning of the sector
*      call [sendln] to output sector text line
*      for each sector of plot line do
*          clear all bits of plot line buffer
*      call [sendln] to output blank line
*      add fiducial parameter mnemonic to print line (buffer2)
*      call [sendln] to output print line text
*      copy initial fiducial value to print line text
*      call [sendln] to output print line text
*      if fiducial parm. type is hex then

```

```

*           call [txthex] to convert text to integer value
*       else
*           call [txtdec] to convert text to integer value
*       add scale marks to plot line buffer
*       set fiducial marker print flag to TRUE
*       if no errors then
*           begin
*               set old fiducial parameter variable to new value
*               if fiducial parm. type is hex then
*                   call [hextxt] to add value to print line (buffer2)
*               else
*                   call [dectxt] to add value to print line (buffer2)
*               set fiducial counter to 1
*               set plot line counter to 0
*           end
*       end
*   end
* end
* end
* if no errors then
*   begin
*     set plot parameter pointer to beginning of table
*     while no errors and plot parm. pointer is not at end of table do
*       if plot parameter is defined for table entry then
*         begin
*           if first table entry or plot type is not succ. diff. then
*             begin
*               if plot parameter type is hex then
*                 call [txthex] to extract value from data block
*               else
*                 call [txtdec] to extract value from data block
*             end
*           if no errors and successive difference plot then
*             begin
*               if s. d. variable is greater or equal to 7 then
*                 set sixth succ. diff. to new value -
*                   zero succ. diff. -
*                   first succ. diff. - second succ. diff. -
*                   third succ. diff. - fourth succ. diff. -
*                   fifth succ. diff.
*               else if s. d. variable is greater or equal to 6 then
*                 set fifth succ. diff. to new value -
*                   zero succ. diff. -
*                   first succ. diff. - second succ. diff. -
*                   third succ. diff. - fourth succ. diff.
*               else if s. d. variable is greater or equal to 5 then
*                 set fourth succ. diff. to new value -

```

```

*           zero succ. diff. -
*           first succ. diff. - second succ. diff. -
*           third succ. diff.
*       else if s.d. variable is greater or equal to 4 then
*           set third succ. diff. to new value -
*           zero succ. diff. -
*           first succ. diff. - second succ. diff.
*       else if s. d. variable is greater or equal to 3 then
*           set second succ. diff. to new value -
*           zero succ. diff. -
*           first succ. diff.
*       else if s. d. variable is greater or equal to 2 then
*           set first succ. diff. to new value -
*           zero succ. diff.
*       set zero succ. diff. to new value
*       if succ. diff. variable is not 7 then
*           increment succ. diff. variable
*       clear memory value for plot parameter
*       end
*   end
* else if plot is successive difference then
*   begin
*     if first character of plot parm. code is not 'S' then
*       call [dsperr] with "invalid parameter code"
*     else
*       case second character of code of
*         'G': set value to zero succ. diff. value
*         'H': set value to first succ. diff. value
*         'I': set value to second succ. diff. value
*         'J': set value to third succ. diff. value
*         'K': set value to fourth succ. diff. value
*         'L': set value to fifth succ. diff. value
*         'M': set value to sixth succ. diff. value
*       default:
*         call [dsperr] with "invalid parameter code"
*     end
*   end
* if no errors and plot table index is not beginning entry or
*   plot table index is beginning entry and plot
*   parameter is not successive difference then
*   begin
*     divide value by plot scale
*     if value is negative then
*       add scale to value
*     if plot type is successive difference then
*       if value is greater than half the plot scale then

```

```

*           subtract half the plot scale from the value
*       else
*           add half the plot scale to the value
*       multiply value by number of pixels in total making up
*           the plot zone (end sector less start sector)
*       divide value by plot scale
*       save this value at table index of memory array
*       call [ipower] to set bit to: 2 to the power of
*           (one less than the number of bits per plot sector -
*           value divided by number of bits per plot sector)
*       set bit in plot line:
*           exclusive 'or' the bit at location 'scaled
*           parameter value divided by number of bits per plot
*           sector + start sector address times the number of
*           bits per plot sector'
*       end
*   end
*   if no errors then
*       begin
*           if fiducial parameter type is hex then
*               call [txthex] to convert text to an integer value
*           else
*               call [txtdec] to convert text to an integer value
*           if no errors and new value - old fiducial parm. value is
*               greater or equal to fiducial interval then
*               begin
*                   add fiducial interval to old fiducial value
*                   if fiducial scale marks counter is 0
*                       (time to print actual value) then
*                           begin
*                               if fiducial parameter type is hex then
*                                   call [hextxt] to convert fid. parm. to print line
*                                   text (buffer2)
*                               else
*                                   call [dectxt] to convert fid. parm. to print line
*                                   text (buffer2)
*                               end
*                           set scale markers in plot line buffer
*                           set fiducial print flag to TRUE
*                           if fiducial scale mark counter greater than 9 then
*                               set counter to 0
*                           else
*                               increment counter
*                           end
*                       if a full print/plot line combination has been completed
*                           and no errors then

```

2100-12-002

```
*      begin
*      if 5 print/plot line combinations have been output then
*          call [iowrit] to output mnemonic (of plot parms) line bfr
*      else if 5 print/plot line combinations and
*          the mnemonic line text have been output then
*          call [iowrit] to output scale value line buffer
*      else if 5 print/plot line combination and the scale value line
*          text have been output then
*          call [iowrit] to output plot parm. units line text (bfr2)
*      else
*          call [iowrit] to output the print line text (buffer2)
*      clear print line buffer
*      flag end of buffer with carriage return and null
*      end
*  if no errors then
*      begin
*          call [iowrit] to output plot line
*          if fiducial marker flag is set then
*              begin
*                  exclusive or out the scale markers from the plot buffer
*                  set fiducial marker flag to FALSE
*              end
*          if plot type is succ. diff. then
*              set memory array index to 0
*          else
*              set memory array index to -1
*          while memory array index is not at end of table do
*              begin
*                  call [ipower] to set bit based on parm. value (see above)
*                  exclusive 'or' out the bit for the parm. in the plot buffer
*              end
*          for the number of vertical space lines required do
*              begin
*                  increment plot line counter
*                  if no errors and first line of print/plot combination then
*                      begin
*                          call [iowrit] to output print line
*                          clear print line buffer
*                          mark end of buffer with null and carriage return chars.
*                      end
*                  call [iowrit] to output plot line buffer
*                  end
*              increment plot line counter
*          end
*      if record count is 9 then
*          reset record counter to 0 (time to read new block of data)
```

2100-12-002

```
*           else
*             increment record counter
*           end
*         end
*       if no errors then
*         call [dsperr] with "... plotting complete"
*         call [ioclos] to close the source device
*         call [ioclos] to close the Versatec Printer/Plotter
*         return status value
*       end
*
```

*****/

2100-12-002

5.3.1 Integer Exponentiation

```

/*****
*
* Name:
*   ipower
*
* Function:
*   Calculates integer exponentiation
*
* Synopsis:
*   ipower(x,n)
*
* Input Parameters:
*   x:   base
*   n:   exponent
*
* Output Parameters:
*
* Return Values:
*   p:   x to the power n
*
* Functions Referenced:
*
* Global Variables Referenced:
*
* Description:
*   Multiplies x by itself n times
*
* Algorithm:
*   begin
*       set return value to 1
*       for n times do
*           multiply p by x
*       return p
*   end
*
*****/
```

2100-12-002

5.3.2 Create a Simultaneous Print/Plot Line on the Versatec

```

/*****
*
* Name:
*     sendln
*
* Function:
*     prints an entire raster line on output device
*
* Synopsis:
*     sendln(df,prt_line,plt_line)
*
* Input Parameters:
*     df:          file identifier for destination device
*     prt_line:    ASCII text to be printed
*     plt_line:    plot line to overlay ASCII text
*
* Output Parameters:
*
* Return Values:
*     NULL_ERR = okay
*
* Functions Referenced:
*     iowrit:  write a data block to device
*
* Global Variables Referenced:
*
* Description:
*     Print plot text and required number of plot lines
*
* Algorithm:
*     begin
*         set return value to NULL_ERR
*         call [iowrit] to output print line buffer
*         for required number of plot lines do
*             call [iowrit] to output plot line buffer
*         return status value
*     end
*
*****/
```


2100-12-002

6.0 STACKED PROFILE ROUTINES

6.1 Define Extraction Parameters

```

/*****
*
* Name:
*   stkprf
*
* Function:
*   Defines stacked profile specifications, performs extraction and plotting
*
* Synopsis:
*   stkprf(fileid)
*
* Input Parameters:
*   fileid:  coded portion of file name for display file
*
* Output Parameters:
*
* Return Values:
*   NULL_ERR = okay
*   negative = see [dspscr]
*
* Functions Referenced:
*   chgval:  change a screen value in the Presentation Area
*   dsperr:  display message on Error Line window
*   dspscr:  present menu/display on the Monitor
*   echo:    turn on keyboard echoing
*   incprc:  define included line data
*   inpstr:  prompt and get a string via keyboard
*   movcsr:  move the visible cursor on the Monitor
*   noecho:  turn off keyboard echoing
*   system:  VENIX O/S call
*   xtrchk:  verify stacked profile extraction specifications
*   xtrprc:  perform extraction of stacked profile data
*   wgetch:  get a character from the keyboard through the window
*   wvalue:  write changed values of Presentation Area back to value file
*
* Global Variables Referenced:
*   prompt_line:  Prompt Line window of display
*   screen:       attributes and description of Presentation Area
*
*****/
```

2100-12-002

```
* Description:
*   Presents the Stacked Profile Specifications Menu and accepts and validates
*   Operator input until a 'P' in the Input Line at which time the extraction
*   process will commence. When extraction is complete, plotting will
*   commence.
*
* Algorithm:
*   begin
*       set return value to NULL_ERR
*       call [dspscr] to display menu
*       if error from [dspscr] then
*           return status value
*       call [noecho] to turn off keyboard echo
*       set extraction performed flag to FALSE
*       set input response to 0
*       while input response is not ((CTRL Z and no errors) or CTRL C) do
*           begin
*               call [wgetch] to get keyboard input
*               call [movcsr] to move cursor if requested
*               if return status from [movcsr] is NOT_CRSR_STS then
*                   begin
*                       call [chgval] to change the screen value
*                       if return status from [chgval] is INP_LINE_STS then
*                           begin
*                               call [xtrchk] to verify parameters to be extracted
*                               if no errors from [xtrchk] and input is 'P' then
*                                   begin
*                                       call [xtrprc] to extract stacked profile data
*                                       if no error from [xtrprc] then
*                                           begin
*                                               set extraction performed flag to TRUE
*                                               call [wvalue] to temporarily store screen values
*                                               call [incprc] to define included line data
*                                               call [dspscr] to redisplay last screen
*                                           end
*                                       end
*                                   end
*                               end
*                           end
*                       end
*                   end
*               end
*           end
*       if input response is CTRL Z then
*           call [wvalue] to save updated values
*       call [echo] to turn on keyboard echo
*       if extraction performed then
*           begin
*               call [inpstr] to prompt Operator: "Delete Extraction Data?"
```

2100-12-002

```
*           if input is "yes" then
*             call [system] to remove all files in the "tmp" subdirectory
*             end
*           return status value
*         end
*
*/
```

2100-12-002

6.2 Verify Extraction Parameters

```

/*****
*
* Name:
*   xtrchk
*
* Function:
*   Verifies stacked profile extraction specifications
*
* Synopsis:
*   xtrchk(stk_tbl)
*
* Input Parameters:
*
* Output Parameters:
*   stk_tbl:  table containing parameters, scales, and positions to be
*             plotted
*
* Return Values:
*   NULL_ERR      = okay
*   INV_CHAR_ERR  = invalid character for defining logical unit
*   INV_P_CODE_ERR = invalid parameter code
*
* Functions Referenced:
*   dsperr:  display message on Error Line window
*   prmtch:  find parameter code match in parameter table
*
* Global Variables Referenced:
*   scr:  attributes and descriptions of fields on screen
*
* Description:
*   Verifies that the extraction parameters set up by the Operator via the
*   keyboard and screen are valid
*
* Algorithm:
*   begin
*       set return value to NULL_ERR
*       set field pointer to field to right of Input Line
*       while no errors and field not a single character long do
*           begin
*               call [prmtch] to verify code
*               if error returned from [prmtch] then
*                   call [dsperr] to display message "invalid parameter code"
*****/
```

2100-12-002

```
*      else
*      begin
*      save index into recording parameter table in stack table
*      move to next field at right
*      end
*      end
*      if no errors then
*      begin
*      if field is source lun then
*      begin
*      if lun is smaller than smallest valid lun or
*      larger than largest valid lun then
*      call [dsperr] to display message "invalid character"
*      else
*      save lun in stack table
*      end
*      end
*      end
*      return status value
*      end
*
*****/
```

6.2.1 Extract Parameters from Data

```

/*****
*
* Name:
*   xtrprc
*
* Function:
*   Extracts stacked profile data into separate files
*
* Synopsis:
*   xtrprc(stk_tbl)
*
* Input Parameters:
*   stk_tbl:  contains relevant parameters for creating stacked profile
*
* Output Parameter's:
*
* Return Values:
*   negative = error encountered
*   DONE_SRT_STK_STS = process performed to completion
*   INV_DEV_ERR      = invalid source device for obtaining profile data
*
* Functions Referenced:
*   close:  close a file for access
*   creat:  create a new file on the IBM-AT disk
*   dsperr: display a message on the Error Line window
*   ioclos: close a device for access
*   ioopen: open a device for access
*   ioread: read a data buffer from device
*   sserch: find parameter code in data buffer1
*   txtdec: convert text to decimal value
*   txthex: convert text to hexadecimal value
*   write:  write a data buffer to temporary file
*
* Global Variables Referenced:
*   buffer1: data block storage area
*   buffer2: data block storage area
*   rec_prm: table of valid recording parameters
*   lun_tbl: table mapping logical units to physical devices
*
* Description:
*   Extract all data for each traverse line number for Stacked Profile
*   plot into separate files.

```

```

*
* Algorithm:
*   begin
*       set return value to NULL_ERR
*       if source lun magnetic tape or hard disk then
*           call [ioopen] to open source device
*       else if source lun not NUL then
*           call [dsperr] with message "invalid source lun"
*       else
*           begin
*               call [dsperr] with "extraction commencing ..."
*               call [open] to open stack table data type file
*               if error from [open] then
*                   call [dsperr] with message "error at source device"
*               else
*                   begin
*                       call [read] to read stack table data
*                       if error from read then
*                           call [dsperr] with message "error at source device"
*                       else
*                           begin
*                               get stack table from data read
*                               call [dsperr] with message "extraction complete"
*                           end
*                       call [close] to close stack table data file
*                   end
*               end
*           if errors or status is "extraction complete" then
*               return status
*           call [dsperr] with "extraction commencing ..."
*           set current line number to NULL_PTR
*           set reverse flag to TRUE
*           set block length to maximum
*           initialize filename template
*           initialize line field of line table of stack table to negative values
*           set done flag to FALSE
*           while no errors and not done do
*               begin
*                   call [ioread] to get a data buffer
*                   if error status or end of source device then
*                       set done flag to TRUE
*                   else
*                       begin
*                           if first line then
*                               begin
*                                   * find size of recording buffer (end marked by null character)

```

```

*      call [sserch] to find line number parameter in data buffer
*      if error from [sserch] then
*          call [dsperr] with message "missing parameter code"
*      else
*          begin
*              set current record length to 0
*              set new record length to 0
*              while no errors and not at end of extraction parameter
*                  table do
*                  begin
*                      if extraction parameter is not NULL then
*                          begin
*                              call [sserch] to find parameter in data buffer
*                              if error from [sserch] then
*                                  call [dsperr] with "missing parameter code"
*                              else
*                                  begin
*                                      if current record length is zero then
*                                          begin
*                                              call [sserch] to find second occurrence of
*                                                  parameter in data buffer
*                                              subtract two locations and add 2 for the
*                                                  parm code to get data record length
*                                              calculate header length as buffer length less
*                                                  10 * current record length
*                                          end
*                                      save new record position in stack table
*                                      add length of parameter to new record length
*                                  end
*                              end
*                          end
*                      calculate length of new buffer as new record length * 10
*                          + header length
*                      add a carriage return to end of new buffer
*                  end
*              end
*          call [sserch] to find date parameter in data buffer
*          if not found then
*              set date in stack table to 0
*          else
*              begin
*                  call [txtdec] to convert date to integer
*                  save value in stack table
*              end
*          end
*      end
*      if no errors then

```



```

*      call [txthex] to get line number from data buffer
*
*      if no errors and
*          line number not same as current line number then
*      begin
*          if current line number not negative then
*              begin
*                  set include flag of table for line to -1
*                  call [close] to close old line file
*                  save current line number in stack table
*                  save line direction in stack table
*                  increment index to next entry in line table of stk tbl
*              end
*          if reverse flag is TRUE then
*              set reverse flag to FALSE
*          else
*              set reverse flag to TRUE
*          set current line number to data buffer's line number
*          add line number to filename template
*          call [creat] to open a file with line number name
*          if error from [creat] then
*              call [dsperr] with message "error at destination device"
*          else
*              set block count for new line to 0
*          end
*      if no errors then
*          begin
*              increment block count for traverse line
*              copy date and line number from data blk to temporary blk
*              for each of ten records do
*                  for each element of the extraction table do
*                      begin
*                          if a parameter is defined do
*                              for each character/digit of parameter do
*                                  copy parameter to temporary buffer
*                              end
*                          call [write] to write temporary buffer
*                          if number of chars written not same as buffer length then
*                              call [dsperr] with "error at destination device"
*                          end
*                      end
*                  end
*              end
*          end
*      if line number not negative then
*          begin
*              set include flag of table for line to -1
*              call [close] to to close the last line file
*              save line number in stack table

```

2100-12-002

```
*      save line direction in stack table
*      end
*      call [ioclos] to close source device
*      if status is END_SRC_DEV_STS then
*          begin
*              set last byte of stack table to marker character ('*')
*              call [creat] to open file for stack table data structure
*              call [write] to write data struct to IBM-AT disk
*              call [close] to close file
*              call [dsperr] with "extraction complete"
*          end
*      return status value
*      end
*
*****/
```

6.3 Define Inclusion Flight Lines

```

/*****
*
* Name:
*   incprc
*
* Function:
*   Defines the line data that is to be included during Stacked Profiling
*
* Synopsis:
*   incprc(fileid,stk_tbl)
*
* Input Parameters:
*   fileid:   coded portion of file name for display file
*   stk_tbl:  stacked profile plotting specifications table
*
* Output Parameters:
*
* Return Values:
*   NULL_ERR = okay
*   negative = see [dspscr]
*
* Functions Referenced:
*   chgval:   change a screen value in the Presentation Area
*   dspscr:   present menu/display on the Monitor
*   incchk:   check inclusion specifications
*   movcsr:   move the visible cursor on the Monitor
*   stkprc:   define stacked profile plot specifications
*   wgetch:   get a character from the keyboard through the window
*   wvalue:   write changed values of Presentation Area back to value file
*
* Global Variables Referenced:
*   prompt_line: Prompt Line window of display
*   screen:       attributes and description of Presentation Area
*
* Description:
*   Presents the Stacked Profile Exclusion Specifications Menu and accepts and
*   validates Operator input until a 'P' in the Input Line at which time
*   the plotting specifications display will be presented.
*
* Algorithm:
*   begin
*       set return value to NULL_ERR

```

```

*      call [dspscr] to display menu
*      if error from [dspscr] then
*          return status value
*      set input response to 0
*      while input response is not ((CTRL Z and no errors) or CTRL C) do
*          begin
*              call [wgetch] to get keyboard input
*              call [movcsr] to move cursor if requested
*              if return status from [movcsr] is NOT_CRSR_STS then
*                  begin
*                      call [chgval] to change the screen value
*                      if return status from [chgval] is INP_LINE_STS then
*                          begin
*                              call [incchk] to verify inclusion specifications
*                              if no errors from [stkchk] and key is 'P' then
*                                  begin
*                                      call [wvalue] to temp. save screen values
*                                      call [stkprc] to define stacked profile plot specs.
*                                      call [dspscr] to redisplay menu
*                                  end
*                              end
*                          end
*                      end
*                  end
*              end
*          end
*      if input response is CTRL Z then
*          call [wvalue] to save updated values
*      return status value
*  end

```

```

*****/

```

6.3.1 Verify Inclusion Flight Lines

```

/*****
*
* Name:
*   incchk
*
* Function:
*   Verifies stacked profile plot inclusion parameters
*
* Synopsis:
*   incchk(stk_tbl)
*
* Input Parameters:
*
* Output Parameters:
*   stk_tbl:  contains parameters, scale, and positions for plotting
*
* Return Values:
*   NULL_ERR      = okay
*   INV_LINE_NBR_ERR = invalid line number
*   INV_RESPONSE_ERR = invalid response
*
* Functions Referenced:
*   dsperr:  display message on Error Line window
*   txthex:  convert text to hexadecimal integer
*
* Global Variables Referenced:
*   scr:     attributes and descriptions of fields on screen
*
* Description:
*   Verifies that the stacked profile inclusion specifications set up by the
*   Operator via the keyboard and screen are valid
*
* Algorithm:
*   begin
*       set return value to NULL_ERR
*       set field pointer to right of Input Line
*       set last pointer to NULL pointer
*       while no errors and not at Input Line do
*           begin
*               { line number }
*               get first character of field from screen
*               if first character is a space then

```

```

*      move to the field at the right of the right
*   else
*       begin
*       call [txthex] to convert line number to integer
*       if no error from [txthex] then
*           begin
*           search line table of stack table for the start line
*           if not found then
*               call [dsperr] with "invalid line number"
*           else
*               begin
*               set print line flag in stack table to TRUE
*               if last flag is not NULL then
*                   set next sequential line pointer to last pointer
*               else
*                   set first line pointer to line index
*                   set last pointer to current line index
*               end
*           end
*       end
*       move to next field at right
*       if no errors then
*           begin
*               { flight line orientation }
*           get character from screen
*           if character is '1' then
*               set flag in stack table to 1
*           else if character = '0' then
*               set flag in stack table to 0
*           else if character is space then
*               set flag to extracted orientation
*           else
*               call [dsperr] with message "invalid response"
*           end
*       end
*       move to next field at right
*       end
*   end
*   if no errors then
*       save next sequential line for last pointer to NULL
*   return status value
* end

```

*****/

6.4 Define Stacked Profiling Specifications

```

/*****
*
* Name:
*   stkprc
*
* Function:
*   Defines stacked profile plot specifications and performs plotting
*
* Synopsis:
*   stkprc(fileid,stk_tbl)
*
* Input Parameters:
*   fileid:   coded portion of file name for display file
*   stk_tbl:  stacked profile plotting specifications table
*
* Output Parameters:
*
* Return Values:
*   NULL_ERR = okay
*   negative = see [dspscr]
*
* Functions Referenced:
*   chgval:   change a screen value in the Presentation Area
*   dspscr:   present menu/display on the Monitor
*   movcsr:   move the visible cursor on the Monitor
*   stkchk:   check plotting specifications
*   stkplt:   create stacked profile plot
*   wgetch:   get a character from the keyboard through the window
*   wvalue:   write changed values of Presentation Area back to value file
*
* Global Variables Referenced:
*   prompt_line: Prompt Line window of display
*
* Description:
*   Presents the Stacked Profile Plotting Specifications Menu and accepts and
*   validates Operator input until an <enter> in the Input Line at which time
*   the plotting process will commence.
*
* Algorithm:
*   begin
*       set return value to NULL_ERR
*       call [dspscr] to display menu

```

2100-12-002

```
*      if error from [dspscr] then
*          return status value
*      set input response to 0
*      while input response is not ((CTRL Z and no errors) or CTRL C) do
*          begin
*              call [wgetch] to get keyboard input
*              call [movcsr] to move cursor if requested
*              if return status from [movcsr] is NOT_CRSR_STS then
*                  begin
*                      call [chgval] to change the screen value
*                      if return status from [chgval] is INP_LINE_STS then
*                          begin
*                              call [stkchk] to verify plot specifications
*                              if no errors from [stkchk] and key is 'P' then
*                                  call [stkplt] to create stacked profile plot
*                              end
*                          end
*                      end
*                  end
*              end
*          end
*      if input response is CTRL Z then
*          call [wvalue] to save updated values
*      return status value
*  end
*
*****/
```


6.4.1 Verify Stacked Profiling Specifications

```

/*****
*
* Name:
*   stkchk
*
* Function:
*   Verifies stacked profile plot specifications
*
* Synopsis:
*   stkchk(stk_tbl)
*
* Input Parameters:
*
* Output Parameters:
*   stk_tbl:  contains parameters, scale, and positions for plotting
*
* Return Values:
*   NULL_ERR      = okay
*   AMB_SPEC_ERR  = ambiguous specification
*   INV_PLOT_POSN_ERR = invalid plot position
*   INV_P_CODE_ERR = invalid parameter code
*   INV_RESPONSE_ERR = invalid response
*   LIMIT_ERR     = limit error
*   NO_PARM_ERR   = missing parameter
*
* Functions Referenced:
*   dsperr:  display message on Error Line window
*   prmtch:  find parameter code match in parameter table
*   txtdec:  convert text to decimal integer
*
* Global Variables Referenced:
*   scr:     attributes and descriptions of fields on screen
*
* Description:
*   Verifies that the stacked profile plot specifications set up by the
*   Operator via the keyboard and screen are valid
*
* Algorithm:
*   begin
*       set return value to NULL_ERR
*       set field pointer to right of Input Line
*       { profile parameter }

```

```

*      call [prmtch] to match parameter code in rec parm table
*      if error from [prmtch] then
*          call [dsperr] to display message "invalid parameter code"
*      else
*          begin
*              search the sort table to verify that the profile parameter
*                  has indeed been extracted
*              if the parameter has not been extracted then
*                  call [dsperr] with message "missing parameter"
*              else
*                  save extraction index in stack table
*              end
*      if no errors then
*          begin
*              move to next field at right
*                  { free-format }
*              set format pointer to NULL_PTR
*              get character from screen
*              if char 'Y' or 'y' then
*                  begin
*                      set format pointer to "time" index in recording table
*                      set free-format flag in stack table to TRUE
*                  end
*              else if char is not 'N' or 'n' then
*                  call [dsperr] with message "invalid response"
*              end
*      if no errors then
*          begin
*              move to next field at right
*                  { time-adjust }
*              get character from screen
*              if char = 'Y' or 'y' then
*                  begin
*                      if a format parameter has been specified then
*                          call [dsperr] with message "ambiguous specification"
*                      else
*                          begin
*                              set format pointer to "time" index in recording table
*                              set free-format flag in stack table to FALSE
*                          end
*                      end
*                  else if char not 'N' or 'n' then
*                      call [dsperr] with message "invalid response"
*                  end
*      if no errors then
*          begin

```

```

*      move to next field at right
*      { other }
*      call [prmtch] to match parameter code in rec parm table
*      if error from [prmtch] then
*          call [dsperr] to display message "invalid parameter code"
*      else if valid "other" parameter and a format parameter already
*          defined then
*          call [dsperr] with message "ambiguous specification"
*      else if NULL "other" parameter and no format parameter already
*          defined then
*          call [dsperr] with message "missing parameter"
*      else
*          set format pointer to index returned from [prmtch]
*      end
*  if no errors and a format parameter has been specified then
*      begin
*          search extracted parameters in stack table for format parameter
*          if not found then
*              call [dsperr] with message "missing parameter"
*          else
*              save extraction index in stack table
*          end
*  if no errors then
*      begin
*          move to next field at right
*          { format interval }
*          call [txtdec] to convert text to decimal integer
*          if no error then
*              save interval in stack table
*          end
*  if no errors then
*      begin
*          move to next field at right
*          { list line numbers }
*          get character from screen
*          if character is 'Y' or 'y' then
*              set flag in stack table to TRUE
*          else
*              set flag in stack table to FALSE
*          else
*              call [dsperr] with "invalid response"
*          end
*  if no errors then
*      begin
*          move to next field at right
*          { plot parameter scale }

```

```

*      .call [txtdec] to convert scale text to an integer value
*      if no errors from [txtdec] then
*          save value in stack table
*      end
*  if no errors then
*      begin
*          move to next field at right
*          { first plot position -- start coordinate }
*      call [txtdec] to convert text to decimal value
*      if no errors from [txtdec] then
*          if value > maximum number of sectors then
*              call [dsperr] to display message "invalid plot position"
*          else
*              begin
*                  save converted value in stack table
*                  { first plot position -- stop coordinate }
*                  call [txtdec] to convert text to decimal value
*                  if no errors from [txtdec] then
*                      if value > maximum number of sectors
*                          or start value > value then
*                          call [dsperr] to display message "invalid plot position"
*                      else
*                          save converted value in stack table
*                  end
*              end
*          end
*      if no errors then
*          begin
*              move to next field at right
*              { separation }
*              call [txtdec] to convert text to decimal value
*              if no errors from [txtdec] then
*                  if value > maximum number of sectors then
*                      call [dsperr] to display message "invalid plot position"
*                  else
*                      save converted value in stack table
*                  end
*              end
*          if no errors then
*              begin
*                  move to next field at right
*                  { vertical scale }
*                  call [txtdec] to convert text to decimal value
*                  if no errors from [txtdec] then
*                      if value > maximum vertical scale then
*                          call [dsperr] to display message "limit error"
*                      else
*                          save converted value in stack table

```

2100-12-002

```
*      end
*      if no errors then
*          begin
*              move to next field at right
*              { include scale lines }
*              get character from screen
*              if character is 'Y' or 'y' then
*                  set flag in stack table to TRUE
*              else
*                  set flag in stack table to FALSE
*              else
*                  call [dsperr] with "invalid response"
*              end
*          return status value
*      end
*
*****/
```

6.5 Stacked Profiling Process

```

/*****
*
* Name:
*     stkplt
*
* Function:
*     Creates and plots the stacked profiles
*
* Synopsis:
*     stkplt(stk_tbl)
*
* Input Parameters:
*     stk_tbl:  contains relevant parameters for creating stacked profiles
*
* Output Parameters:
*
* Return Values:
*     NULL_ERR      = okay
*
* Functions Referenced:
*     abs:         find integer absolute value
*     close:       close a file for access
*     dectxt:      convert decimal value to text
*     dsperr:      display a message on the Error Line window
*     rdblock:     read a block of profile and format values
*     hextxt:      convert hex value to text
*     ipower:      integer version of exponentiation
*     ioclos:      close a device for access
*     ioopen:      open a device for access
*     iowrit:      write a data block to device
*     lseek:       seek to a location in file
*     open:        open a file for access
*     stkhdr:      print stacked profile header
*
* Global Variables Referenced:
*     buffer1:     data area for print and plot buffers
*     lun_tbl:     table mapping physical devices to logical unit numbers
*     rec_prm:     recording parameter table of mnemonics, lengths etc.
*
* Description:
*     Creates the stacked profile plot on the Versatec printer/plotter.
*
*/

```

```

* Algorithm:
*   begin
*       set return value to NULL_ERR
*       find Versatec plotter/printer in lun table
*       if not found then
*           call [dsperr] with message "invalid destination device"
*       else
*           begin
*               call [ioopen] to open Versatec plotter/printer in SPP mode
*               if error from [ioopen] then
*                   call [dsperr] to display message "cannot open destination device"
*               end
*           if error has been encountered then
*               return status
*           call [dsperr] with "stacked profiling commencing ..."
*           clear plot line buffer (buffer1)
*           clear print line buffer (buffer1)
*           place carriage return and null characters at end of print bufffer
*           call [stkhdr] to print header of profile on Versatec
*           clear filename string
*           copy filename template to filename string
*           set a counter to the number of flight lines
*           set linked line to NULL_PTR
*           set total number of flight lines counter to -1
*           set number of lines to be printed counter to 0
*           while flight line in stack table greater than zero do
*               begin
*                   get flight line number as a value
*                   increment total number of flight lines
*                   if flight line is to be printed then
*                       begin
*                           call [hextxt] to add flight line number to filename
*                           call [open] to open the extraction file for the line data
*                           if vertical scale lines are desired then
*                               begin
*                                   mark beginning of line sector boundary in plot buffer
*                                   mark end of line sector boundary in plot buffer
*                               end
*                           increment number of lines to be printed
*                       end
*                   end
*               clear plot points 'memory' buffer
*               set first block flag to TRUE
*               set done flag to FALSE
*               set fiducial parm. print flag to FALSE
*               clear total records counter

```

```

*      clear fiducial counter
*      clear plot line counter
*      set line list flag to first flight line
*      while no errors and not done do
*          begin
*              if first block of data then
*                  begin
*                      for each line number do
*                          if line is to be printed then
*                              begin
*                                  set record counter to 0
*                                  calculate number of bytes to read
*                                  if the line data is reversed then
*                                      call [lseek] to move to the end of the file
*                                      call [rdbrck] to read a block of data and extract values
*                                  end
*                              end
*                          set line pointer to start line
*                          set temp. pointer to start line
*                          get format value from line data table
*                          while not all lines have been checked do
*                              begin
*                                  if format parameter type is not time-adjusted or free format
*                                      and format value for line is less than start line
*                                      format value or
*                                      type is time-adjusted or free format and
*                                      and number of blocks for line is greater than that
*                                      for start line then
*                                      begin
*                                          save format value for line as reference value
*                                          save flight line table index as reference line
*                                      end
*                                  move to next line
*                              end
*                          set first block flag to FALSE
*                      end
*                  if no errors then
*                      begin
*                          set parameter pointer to zero
*                          if first plot line to be formed or
*                              the difference between the format reference value and
*                              the fiducial value is greater than format interval then
*                              begin
*                                  if fiducial value is zero then
*                                      set fiducial value to format reference value
*                                  else if fiducial value is growing smaller than

```



```

*          subtract fiducial interval from fiducial value
*      else
*          add fiducial interval to fiducial value
*      set fiducial scale marker in plot line at each end
*      set fiducial print flag to TRUE
*      if fiducial counter is zero then
*          begin
*              if format parameter type is hex then
*                  call [hextxt] to add format value to print buffer
*              else
*                  call [dectxt] to add format value to print buffer
*              end
*          if fiducial counter is less than 9 then
*              increment fiducial counter
*          else
*              set fiducial counter to zero
*          end
*      set line pointer to start line
*      while not all flight lines processed do
*          begin
*              if the data file for the flight line is closed or
*                  format type is free and line record count
*                      times reference line block count divided by
*                      line block count is not equal to total
*                      records processed so far or
*                  format type is 'other' and line value is greater than
*                      format reference value then
*                  set memory value for line to NULL_PTR (ie. no plot point)
*              else
*                  begin
*                      get profile value from flight line table
*                      scale value
*                      if value is negative then
*                          add scale to value (full scale wraparound)
*                      convert value to respective plot sector
*                          { value * (end - start sectors + 1)
*                            * no. of bits per char / scale }
*                      save value in memory array for line
*                      call [lpower] to convert sector to bit
*                      add bit to plot line at correct sector location
*                          { (sector / no. bits per char. + start sector * number
*                            bits per char. + parm pointer * separation * no bits
*                            per char.) mod (max number of sector * no. bits
*                            per char) }
*                      increment record count for the line
*                      if record count indicates an even 10 records processed then

```

```

*           begin
*           if record count equals 10 times the block count then
*               begin
*                   call [close] to close the line data file
*                   decrement the flight line counter
*                   if flight line counter equals zero then
*                       set done flag to TRUE
*                   end
*               else
*                   begin
*                       calculate number of bytes to read
*                       if the flight line data is reversed then
*                           call [lseek] to back up two blocks
*                       call [rdblk] to read new format and profile values
*                   end
*               end
*           end
*           increment parameter pointer
*           move to next flight line
*           end
*       if format value not equal to next value from data block then
*           if format type is 'other' then
*               add format interval to format reference value
*           else
*               get new format reference value from reference line data
*           increment total records processed
*       end
*   if no errors and a print line is required then
*       begin
*           if list lines flag set and not all lines have been listed and
*               the desired number of plot lines have been left above
*               as spaces then
*               call [hextxt] to convert line number to text
*           call [iowrit] to output print buffer
*           blank out print buffer
*           if a line number has just been printed then
*               begin
*                   move to next line number
*                   set print line to next line number pointer
*               end
*           end
*       if no errors then
*           begin
*               call [iowrit] to output plot line
*               if fiducial print flag set then
*                   begin

```

2100-12-002

```
*          blank out fiducial scale mark in plot buffer
*          clear fiducial print flag
*          end
*      for each element in memory array do
*          if memory value is NULL_PTR then
*              return memory value to zero
*          else
*              call [ipower] to exclusive 'or' out the correct bit in
*                  the plot buffer for the value plotted
*          for the correct number of vertical spaces do
*              begin
*                  increment plot line counter
*                  if plot line counter indicates a print line is required then
*                      begin
*                          call [iowrit] to output print line
*                          clear print line buffer
*                          place carriage returns at end of buffer
*                      end
*                  call [iowrit] to output a blank plot line
*                  end
*              increment plot line counter
*          end
*      end
*  if done and no errors then
*      call [dsperr] with message "stack profiling complete"
*  if plot line counter indicates not the correct number of plot lines
*      have been output for the previous print line generated then
*      call [iowrit] to output blank plot line
*  call [ioclos] to close the versatec printer/plotter
*  return status value
*  end
*
*****/
```

6.5.1 Obtain a Block of Preprocessed Data

```

/*****
*
*   Name:
*       rdblock
*
*   Function:
*       Reads stacked profile data block and returns desired parameter values
*
*   Synopsis:
*       rdblock(i,len,stk_tbl,f_ln)
*
*   Input Parameters:
*       i:           pointer to line in stack table for which values are being read
*       len:         length of block to be read from file
*       stk_tbl:     table containing parameters, scale and positions for plotting
*
*   Output Parameters:
*       f_ln:        format and profile values read from the block of data
*
*   Return Values:
*       NULL_ERR     = okay
*       SRC_DEV_ERR  = could not read data from traverse line files
*       negative     = see [txtdec], [txthex]
*
*   Functions Referenced:
*       read:        low level I/O to read a device
*       txtdec:      convert text to decimal integer
*       txthex:      convert text to hexadecimal integer
*
*   Global Variables Referenced:
*       buffer2:     data block storage area
*       rec_prm:     recording parameter table of mnemonics, lengths, types etc.
*
*   Description:
*       Reads a block of data from the temporary line file and returns the
*       format and profile values for each record of the block
*
*   Algorithm:
*       begin
*           set return status to NULL_ERR
*           call [read] to get the block of data from line file
*           if length of block read is not equal to length desired then

```

2100-12-002

```
*      call [dsperr] with message "error at source device"
*
*  else
*      begin
*      reset return status to NULL_ERR
*      while no errors and not all ten records processed do
*          begin
*              if format parameter type is hex then
*                  call [txthex] to convert text to integer
*              else
*                  call [txtdec] to convert text to integer
*              if no errors then
*                  begin
*                      if format value is negative then
*                          make positive
*                      if profile parameter type is hex then
*                          call [txthex] to convert text to integer
*                      else
*                          call [txtdec] to convert text to integer
*                  end
*              end
*          end
*      return status value
*  end
*
*****/
```

6.5.2 Create the Stacked Profile Header

```

/*****
*
* Name:
*   stkhdr
*
* Function:
*   Creates and plots the stacked profiles
*
* Synopsis:
*   stkhdr(df,stk_tbl)
*
* Input Parameters:
*   df:      file identifier for destination device
*   stk_tbl: contains relevant parameters for creating stacked profiles
*
* Output Parameters:
*
* Return Values:
*   NULL_ERR = okay
*   negative = see [dectxt], [sendln]
*
* Functions Referenced:
*   dectxt:  convert decimal value to text
*   sendln: send a complete raster scan to output device
*
* Global Variables Referenced:
*   buffer1: data block area for printing and plotting buffers
*   rec_prm: recording parameter table of mnemonics, lengths etc.
*
* Description:
*   Print stacked profile plot header
*
* Algorithm:
*   begin
*       set return value to NULL_ERR
*       clear plot line buffer
*       clear print line buffer
*       add carriage return and null characters to end of print buffer
*       call [sendln] to raster print/plot ten blank lines at top of plot
*       copy header template, part I to print buffer
*       if format parameter is "time" then
*           begin

```

```

*         if format type is 'free' then
*             copy text 'free' to header
*         else
*             copy text 'time-adjusted' to header
*         end
*     else
*         copy format parameter mnemonic to header
*         call [dectxt] to copy format interval to header
*         copy units to header
*         call [sendln] to print header, part I
*         clear print buffer
*         call [sendln] to print a blank line
*         copy header template, part II to print buffer
*         if a date was found in the data then
*             begin
*                 call [dectxt] to add day to header
*                 call [dectxt] to add year to header
*             end
*         call [sendln] to print header, part II
*         clear print buffer
*         call [sendln] to print a blank line
*         copy header template, part III to print buffer
*         copy profile parameter mnemonic to header
*         call [dectxt] to add profile parameter scale to header
*         copy units of scale to header
*         call [sendln] to print header, part III
*         clear print buffer
*         call [sendln] to print a blank line
*         copy header template, part IV to print buffer
*         call [dectxt] to add vertical scale to header
*         call [dextxt] to add separation to header
*         call [sendln] to print header, part IV
*         clear print buffer
*         call [sendln] to print a blank line
*         call [sendln] to print a blank line
*         for each ten unit graduation marking sector positions on plot do
*             add digit to print buffer in this location
*         for each sector location do
*             add a mark in plot line buffer
*         call [sendln] to print this marking scale under header
*         clear print line buffer
*         clear plot line buffer
*         call [sendln] to print a blank line
*         add format parameter mnemonic to print buffer
*         call [sendln] to print mnemonic
*         return status value

```

2100-12-002

* end

*

*****/

7.0 GRADIENT PARAMETER CREATION ROUTINES

7.1 Define Gradient Parameter Specifications

```

/*****
*
*   Name:
*       grdspc
*
*   Function:
*       Defines gradient parameter creation specifications and performs process
*
*   Synopsis:
*       grdspc(fileid)
*
*   Input Parameters:
*       fileid:  coded portion of file name for display file
*
*   Output Parameters:
*
*   Return Values:
*       NULL_ERR = okay
*       negative = error encountered
*
*   Functions Referenced:
*       chgval:   change a screen value in the Presentation Area
*       dmpgrd:   perform gradient parameter creation process
*       dsperr:   display message on Error Line window
*       dspscr:   present menu/display on the Monitor
*       echo:     turn on keyboard echoing
*       grdchk:   verify the gradient parameter specifications defined by Operator
*       movcsr:   move the visible cursor on the Monitor
*       noecho:   turn off keyboard echoing
*       wgetch:   get a character from the keyboard through the window
*       wvalue:   write changed values of Presentation Area back to value file
*
*   Global Variables Referenced:
*       input_line:  Input Line window of display
*       screen:      attributes and description of Presentation Area
*
*   Description:
*       Presents the Gradient Creation Menu and accepts and validates
*       Operator input until a key is depressed while in the Input Line at

```

```

*      which time the gradient creation process will commence.
*
* Algorithm:
*   begin
*       set return value to NULL_ERR
*       call [dspscr] to display menu
*       if error from [dspscr] then
*           return status value
*       call [noecho] to turn off keyboard echo
*       set input response to 0
*       while input response is not ((CTRL Z and no errors) or CTRL C) do
*           begin
*               call [wgetch] to get keyboard input
*               call [movcsr] to move cursor if requested
*               if status from [movcsr] is NOT_CRSR_STS or input is CTRL Z then
*                   begin
*                       call [chgval] to change the screen value
*                       if status from [chgval] is INP_LINE_STS or input is CTRL Z then
*                           begin
*                               call [grdchk] to verify the gradient parameters specified
*                               if no errors from [grdchk] and input is not CTRL Z then
*                                   call [dmpgrd] to perform creation process
*                               end
*                           end
*                       end
*                   end
*               if input response is CTRL Z then
*                   call [wvalue] to save updated values
*                   call [echo] to turn on keyboard echo
*                   return status value
*               end
*           end
*       end
*
* *****/

```

7.2 Verify Gradient Parameter Specifications

```

/*****
*
* Name:
*   grdchk
*
* Function:
*   Verifies the gradient parameter creation specifications
*
* Synopsis:
*   grdchk(grd_tbl)
*
* Input Parameters:
*   grd_tbl:  contains necessary data to create gradient parameters
*
* Output Parameters:
*
* Return Values:
*   NULL_ERR      = okay
*   INV_CHAR_ERR  = invalid character used as logical unit number
*   INV_DEV_ERR   = invalid device
*   INV_P_CODE_ERR = invalid parameter code
*   NO_PARM_ERR   = missing or invalid number of parameters
*   NON_UNIQ_DEV_ERR = source and destination devices are the same
*
* Functions Referenced:
*   dsperr:  display message on Error Line of display
*   prmtch:  find parameter code match in parameter table
*
* Global Variables Referenced:
*   lun_tbl:  table mapping physical devices to logical unit numbers
*   rec_prm:  recording parameters, codes, lengths, etc
*   scr:      attributes and description of data fields on screen
*
* Description:
*   Verifies the gradient parameter specifications defined on the Monitor,
*   by the Operator
*
* Algorithm:
*   begin
*       set return value to NULL_ERR
*       set current field to right of Input Line
*       while no errors and field length is longer than one character do

```

```

*      begin
*      call [prmtch] to match the field value against rec parm table
*      if error returned from [prmtch] then
*          call [dsperr] to display message "invalid parameter code"
*      else
*          begin
*              save index in gradient parameter creation table
*              move to next field at right of current field
*          end
*      end
*      if no error encountered then
*          begin
*              count number of consecutive parameters defined on screen
*              if no parms in grad table or number of parms is not even then
*                  call [dsperr] to display message "missing parameter"
*              else
*                  begin
*                      initialize pair table
*                      initialize quad table
*                      set additional record length variable in grad table to 0
*                      set pair flag to 0
*                      set quad flag to 0
*                      for 0 to end of gradient parameter table & while no errors do
*                          begin
*                              if pair flag is 1 then
*                                  begin
*                                      set pair flag to 0
*                                      set parameter code to pair table entry based on pair flag
*                                      call [prmtch] to find parm code in parameter table
*                                      if error from [prmtch] then
*                                          call [dsperr] with message "missing parameter code"
*                                      else
*                                          begin
*                                              save parameter table index in grad table
*                                              save record length as data block pointer for parm
*                                              add length of parm to record length
*                                          end
*                                      end
*                                  else
*                                      set pair flag to 1
*                                  end
*                              if no errors then
*                                  begin
*                                      if quad flag is 3 then
*                                          begin
*                                              set quad flag to 0
*                                              set parameter code to quad table entry based on quad flg

```

```

*           call [prmtch] to find parm code in parameter table
*       if error from [prmtch] then
*           call [dsperr] with message "missing parameter code"
*       else
*           begin
*               save parameter table index in grad table
*               save record length as data block pointer for parm
*               add length of parm to record length
*           end
*       end
*   else
*       increment quad flag
*   end
* end
* end
* end
* if no errors then
*     begin
*         get source lun from display text
*         if source lun less than or equal to 0 or greater than max. lun then
*             call [dsperr] to display message "invalid character"
*         else if source lun not one of MTO, MT1, HDO or HD1 then
*             call [dsperr] with "invalid device"
*         else
*             save source lun in grad table
*         end
*     if no errors then
*         begin
*             get destination lun from display text
*             if dest. lun less than or equal to 0 or greater than max. lun then
*                 call [dsperr] to display message "invalid character"
*             else if source lun not one of MTO, MT1, HDO, HD1, or VFP or NUL then
*                 call [dsperr] with "invalid device"
*             else if source and destination devices are the same then
*                 call [dsperr] with "nonunique devices"
*             else
*                 save destination lun in grad table
*             end
*         return status value
*     end
*
*****/

```

7.3 Gradient Parameter Creation Process

```

/*****
*
*   Name:
*       dmpgrd
*
*   Function:
*       Create gradient parameters and save in recording blocks
*
*   Synopsis:
*       dmpgrd(grd_tbl)
*
*   Input Parameters:
*       grd_tbl:  contains all necessary data to create gradient parameters
*
*   Output Parameters:
*
*   Return Values:
*       NULL_ERR      = okay
*       NO_P_CODE_ERR = missing parameter code in data block
*       negative       = see [ioread] and [wrtblk]
*
*   Functions Referenced:
*       dectxt:  convert a decimal value to a text string
*       dsperr:  display a message on the Error Line of the Monitor
*       hextxt:  convert a hex value to a text string
*       ioclos:  low level routine to close a device
*       ioopen:  low level routine to open a device
*       ioread:  generalized routine for reading a block of data from a device
*       sserch:  search a data block for a string match
*       txtdec:  convert text string to a decimal value
*       txthex:  convert text string to a hex value
*       wrtblk:  write a block of data to destination device
*
*   Global Variables Referenced:
*       buffer1:  data block storage area
*       buffer2:  data block storage area
*       lun_table: maps logical unit numbers to physical devices
*       rec_prms:  recording parameters and specifications
*
*   Description:
*       Performs gradient parameter calculations and appends these new values
*       to each record of each recording block on the source device before writing

```

```

*      the modified block to the destination device.
*
* Algorithm:
*   begin
*       set return value to NULL_ERR
*       call [iioopen] to open source device
*       if no error from [iioopen] then
*           begin
*               call [iioopen] to open destination device
*               if no error from [iioopen] then
*                   call [ioclos] to close source device
*               end
*           if error encountered then
*               return status value
*           set done flag to FALSE
*           set block length to maximum size
*           count the number of parameters involved in the creation process
*           call [dsperr] with message "gradient parameter creation commencing..."
*           while no error and not done do
*               begin
*                   call [ioread] to read a block of data from source device
*                   if error from [ioread] then
*                       set done flag to TRUE
*                   else
*                       begin
*                           if block length is at maximum (first block) then
*                               begin
*                                   determine the size of the data block
*                                   for each defining rec parm reference in grad table do
*                                       begin
*                                           call [sserch] to find parameter code in data block
*                                           if not found then
*                                               call [dsperr] with message "missing parameter code"
*                                           else
*                                               save index of parm code in grad table
*                                               if the first rec. parm then
*                                                   begin
*                                                       call [sserch] to find the second occurrence
*                                                       subtract the first from the second location
*                                                       save as record length
*                                                       calculate header length
*                                                   end
*                                               increment defining parm. pointer
*                                           end
*                                       end
*                                   end
*                               if no errors then

```

```

*           begin
*           copy block header to new block
*           for each record do
*               begin
*               copy record to new block
*               set defining-parameters pointer (dpp) to 0
*               while no errors and dpp < number of def. parms do
*                   begin
*                   if parameter type is decimal then
*                       call [txtdec] to convert defining parameter
*                   else
*                       call [txthex] to convert defining parameter
*                   if dpp MOD 2 equals 1 then
*                       begin
*                       set parameter value = (dpp - 1)th value - (dpp)th value
*                       if pair gradient parameter type is hexadecimal then
*                           call [hextxt] to convert value and save in new block
*                       else
*                           call [dectxt] to convert value and save in new block
*                       save parameter code in temporary block
*                       end
*                   if dpp MOD 4 equals 3 then
*                       begin
*                       set parameter value = ((dpp - 3)th value +
*                                               (dpp - 2)th value) / 2 -
*                                               ((dpp - 1)th value +
*                                               (dpp)th value) / 2
*                       if quad gradient parameter type is hexadecimal then
*                           call [hextxt] to convert value and save in new block
*                       else
*                           call [dectxt] to convert value and save in new block
*                       save parameter code in temporary block
*                       end
*                   increment dpp
*               end
*           end
*           end
*           call [wrtblk] to save the new block
*           . end
*           call [dsperr] with "gradient parameter creation complete"
*           call [ioclos] to close the source device
*           call [ioclos] to close the destination device
*           return status value
*       end
*
*****/

```


2100-12-002

8.0 EDITING ROUTINES

8.1 Present Editor Menu

```

/*****
*
* Name:
*   edtfnt
*
* Function:
*   Performs batch and screen edits
*
* Synopsis:
*   edtfnt(fileid)
*
* Input Parameters:
*   fileid:  coded portion of file name for display file
*
* Output Parameters:
*
* Return Values:
*   NULL_ERR = okay
*   negative = error encountered
*
* Functions Referenced:
*   addval:   add/subtract a value from a recorded parameter
*   chgval:   change a character in the Presentation Area window
*   delprm:   delete/add a recording parameter
*   dsperr:   display a message on the Error Line window
*   dspscr:   display the Presentation Area and Prompt Line windows on screen
*   echo:     turn on keyboard echoing
*   edtchg:   make batch edit changes
*   inpstr:   get a string from the keyboard
*   movcsr:   move the cursor within the Presentation Area window
*   noecho:   turn off keyboard echoing
*   outblk:   output NNN blocks from source to destination devices
*   rdigit:   replace a digit of a recorded parameter
*   rplace:   replace a value for a recorded parameter
*   scredt:   full-screen editor
*   setlim:   set the edit limits for batch editing
*   spkchk:   check for spikes in data during batch edits
*   wgetch:   get a character from keyboard through window
*   wvalue:   write screen values to disk file

```

```

*
* Global Variables Referenced:
*   input_line:  Input Line window of display
*   lun_tbl:     table mapping logical units to physical devices
*   scr:         screen attributes and values
*
* Description:
*   Presents Editing Menu and allows Operator to perform the various batch
*   and screen editing functions as well as defining source and destination
*   devices
*
* Algorithm:
*   begin
*       set return value to NULL_ERR
*       call [dspscr] to display menu
*       if error from [dspscr] then
*           return status value
*       initialize edit table to "null" specifications
*       call [noecho] to turn off keyboard echo
*       set input response to 0
*       while input response is not ((CTRL Z and no errors) or CTRL C) do
*           begin
*               call [wgetch] to get keyboard input
*               call [movcsr] to move cursor if required
*               if status from [movcsr] is NOT_CRSR_STS or input is CTRL Z then
*                   call [chgval] to change the screen value
*               if status from [chgval] is INP_LINE_STS or input is CTRL Z then
*                   begin
*                       get source LUN from field to right of Input Line window
*                       if source lun is less than or equal to 0 or source lun
*                           is greater than maximum lun then
*                           call [dsperr] with "invalid character"
*                       else if lun not magnetic tape or hard disk then
*                           call [dsperr] with "invalid device"
*                       else
*                           begin
*                               save source device in edit table
*                               get destination LUN from field to right of Input Line window
*                               if destination lun is less than or equal to 0 or destination
*                                   lun is greater than maximum lun or source
*                                   call [dsperr] with "invalid character"
*                               else if lun is same as destination lun then
*                                   call [dsperr] with "source device same as destination"
*                               else if lun not magnetic tape or hard disk or null device then
*                                   call [dsperr] with "invalid device"
*                               else

```

```

*           begin
*           save destination device in edit table
*           reset status to NULL_ERR (current INP_LIN_STS)
*           end
*       end
*   if no errors and input response is not CTRL Z then
*       begin
*           call [wvalue] to save temporary screen values
*           case input response of
*               1: call [setlim] to set up edit limits
*               2: call [outblk] to output NNN blocks
*               3: call [addval] to add/subtract a value from a parm
*               4: call [rplace] to replace a parameter value
*               5: call [delprm] to delete/add a parameter
*               6: call [rdigit] to change one digit
*               7: call [spkchk] to check for spikes
*               8: call [echo] to allow keyboard echoing
*                   call [inpstr] to prompt for "are you sure?"
*                   call [noecho] to turn off keyboard echoing
*                   if response is 'y' or 'Y' then
*                       call [edtchg] to make edit changes
*                   9: call [scredit] to use screen editor
*                   default: call [dsperr] to display "invalid option"
*               end
*           call [dspscr] to restore temporary screen values
*       end
*   end
*   if input response is CTRL Z then
*       call [wvalue] to save screen values
*       call [echo] to turn on keyboard echo
*       return status value
*   end
*
*****/

```

8.1.1 Define Batch Editor Limits

```

/*****
*
* Name:
*   setlim
*
* Function:
*   Defines start and stop edit limits and the basis parameter
*
* Synopsis:
*   setlim(fileid,edt_tbl)
*
* Input Parameters:
*   edt_tbl:  contains all parameters and values to be modified for data
*             on source device
*   fileid:   coded portion of file name for display file
*
* Output Parameters:
*
* Return Values:
*   NULL_ERR = okay
*   negative = see [dspscr]
*
* Functions Referenced:
*   chgval:   change a screen value in the Presentation Area
*   dsperr:   display error message on Monitor
*   dspscr:   present menu/display on the Monitor
*   prmtch:   verify a rec parm against those found in the rec parm table
*   movcsr:   move the visible cursor on the Monitor
*   txtdec:   convert text string to a decimal value
*   txthex:   convert text string to a hexadecimal value
*   wgetch:   get a character from the keyboard through the window
*   wvalue:   save updated values in value file
*
* Global Variables Referenced:
*   input_line: Input Line window of display
*   rec_parm:   all recording parameters, lengths, types etc.
*   scr:        attributes and description of Presentation Area
*
* Description:
*   Presents the Edit Limits Display and accepts and validates the keyboard
*   entries and cursor motions made by the Operator to set up the limits,
*   and basis parameter.

```

```

*
* Algorithm:
*   begin
*       set return value to NULL_ERR
*       call [dspscr] to display menu
*       if error from [dspscr] then
*           return status value
*       set input response to 0
*       while input response is not ((CTRL Z and no errors) or CTRL C) do
*           begin
*               call [wgetch] to get keyboard input
*               call [movcsr] to move cursor if required
*               if status from [movcsr] is NOT_CRSR_STS or input is CTRL Z then
*                   call [chgval] to change the screen value
*               if status from [chgval] is INP_LINE_STS or input is CTRL Z then
*                   begin
*                       move to field to right of Input Line window
*                       call [prmtch] to match basis parm with a recording
*                         parameter table entry
*                       if return value from [prmtch] is not zero then
*                           call [dsperr] with message "invalid parameter code"
*                       if no errors then
*                           begin
*                               move to next field at right on screen
*                               if parameter type is hexadecimal then
*                                   call [txthex] to convert start limit to hex value
*                               else
*                                   call [txtdec] to convert start limit to decimal value
*                               end
*                           if no errors then
*                               begin
*                                   move to next field at right on screen
*                                   call [txthex] to convert stop limit to hex value
*                               else
*                                   call [txtdec] to convert stop limit to decimal value
*                               end
*                           end
*                       end
*                   end
*               if input response is CTRL Z then
*                   begin
*                       call [wvalue] to save updated values
*                       save start limit in edit_table
*                       save stop limit in edit_table
*                       save index into rec parm table in edit_table
*                   end
*               return status value

```

2100-12-002

* .end
*
*****/

8.1.2 Dump Data Without Editing

```

/*****
*
*   Name:
*       outblk
*
*   Function:
*       Copy specified number of blocks between media
*
*   Synopsis:
*       outblk(edt_tbl)
*
*   Input Parameters:
*       edt_tbl:  specifications for block editing
*
*   Output Parameters:
*
*   Return Values:
*       NULL_ERR = okay
*       negative = error
*
*   Functions Referenced:
*       asctime:  convert time value to ascii string
*       dsperr:   display message on Error Line of Monitor
*       echo:     allow keyboard echoing
*       fclose:   close file from access
*       fopen:    open a file for access
*       inpstr:   get a text string via keyboard
*       ioread:   read a block of data from a device
*       noecho:   disable keyboard echoing
*       ioclos:   close a device for I/O
*       ioopen:   open a device for input/output
*       printf:   send message to output device
*       system:   make VENIX o/s command call
*       time:     get system time and date
*       txtdec:   convert from text to decimal integer
*       wrtblk:   write a block of data to a device
*
*   Global Variables Referenced:
*       buffer1:   data block storage area
*       lun_tbl:   maps logical unit numbers to physical devices
*
*   Description:

```



```

*      Prompts for number of blocks to be copied, verifies response and
*      . source and destination devices and performs copy.
*
* Algorithm:
*   begin
*       set return value to NULL_ERR
*       call [echo] to turn on keyboard echoing
*       call [inpstr] to get a keyboard integer
*       call [noecho] to turn off keyboard echoing
*       call [txtdec] to convert to input string to integer
*       if no errors do
*           begin
*               call [iopen] to open source device
*               if no errors from [iopen] then
*                   begin
*                       call [iopen] to open destination device
*                       if error from [iopen] then
*                           call [ioclos] to close the source device
*                       end
*                   end
*               if no errors then
*                   begin
*                       set block counter to 0
*                       set done flag to FALSE
*                       while block counter less than NNN and not done do
*                           begin
*                               if block counter is greater or equal to input value then
*                                   set done flag to TRUE
*                               else
*                                   begin
*                                       call [ioread] to get a block of data
*                                       if error from [ioread] then
*                                           set done flag to TRUE
*                                       else if first block then
*                                           calculate true length of data block
*                                       end
*                                       call [wrtblk] to write a block of data
*                                       if no error returned from [iowrit] then
*                                           increment block counter
*                                       end
*                                   end
*                               call [ioclos] to close source device
*                               call [ioclos] to close destination device
*                               call [time] to get the system time and date
*                               call [asctime] to convert to ascii
*                               call [fopen] to open the Star printer
*                               call [printf] with the number of blocks dumped

```

2100-12-002

```
*      call [fclose] to close Star printer
*      end
*      return status value
*      end
*
```

*****/

8.1.3 Add a Value to a Parameter

```

/*****
*
* Name:
*   addval
*
* Function:
*   Add/subtract a value to/from a parameter
*
* Synopsis:
*   addval(fileid,edt_tbl)
*
* Input Parameters:
*   edt_tbl:   contains all parameters and values to be modified for data
*              on source device
*   fileid:    coded portion of file name for display file
*
* Output Parameters:
*
* Return Values:
*   NULL_ERR   = okay
*
* Functions Referenced:
*   chgval:    change a screen value in the Presentation Area
*   dsperr:    display error/status message on Monitor
*   dspscr:    present menu/display on the Monitor
*   movcsr:    move the visible cursor on the Monitor
*   prmtch:    find parameter code match in parameter table
*   txtdec:    convert text string to decimal integer value
*   txthex:    convert text string to hexadecimal number
*   wgetch:    get a character from the keyboard through the window
*   wvalue:    save updated values in value file
*
* Global Variables Referenced:
*   input_line: Input Line window of display
*   rec_prm:    all recording parameters, lengths, types etc.
*   scr:        attributes and description of fields on current screen
*
* Description:
*   Presents the Add/Subtract A Value Display and accepts and validates
*   the keyboard entries and cursor motions made by the Operator to set up
*   the parameters and changes
*
*/

```

```

* Algorithm:
*   begin
*       set return value to NULL_ERR
*       initialize temporary addition table
*       call [dspscr] to display menu
*       if error from [dspscr] then
*           return status value
*       set input response to 0
*       while input response is not ((CTRL Z and no errors) or CTRL C) do
*           begin
*               call [wgetch] to get keyboard input
*               call [movcsr] to move cursor if required
*               if status from [movcsr] is NOT_CRSR_STS or input is CTRL Z then
*                   begin
*                       call [chgval] to change the screen value
*                       if status from [chgval] is INF_LINE_STS or input is CTRL Z then
*                           begin
*                               set field layout table pointer (fltp) to field at right of
*                                   Input Line window
*                               set table pointer to 0
*                               while no errors and ptr less than end of add_val table do
*                                   begin
*                                       call [prmtch] to verify existence of parameter
*                                       if return status from [prmtch] not 0 then
*                                           call [dsperr] with "invalid parameter code"
*                                       else
*                                           begin
*                                               save [prmtch] pointer in a temporary table
*                                               move to next field at right via fltp
*                                               if parameter type is hexadecimal then
*                                                   call [txthex] to convert text to hexadecimal value
*                                               else
*                                                   call [txtdec] to convert text to decimal value
*                                               if no errors then
*                                                   begin
*                                                       save value in a temporary table
*                                                       move to next field at right via fltp
*                                                       increment table pointer
*                                                   end
*                                               end
*                                           end
*                                       end
*                                   end
*                               end
*                           end
*                       end
*                   end
*               if input response is CTRL Z then
*                   begin

```

2100-12-002

```
*          call [wvalue] to save updated values
*          for each temporary table entry do
*              save in edit table
*          end
*      return status value
*  end
*
*****/
```

8.1.4 Replace a Parameter Value With a Constant

```

/*****
*
* Name:
*   rplace
*
* Function:
*   Defines those parameter values to be replaced
*
* Synopsis:
*   rplace(fileid,edt_tbl)
*
* Input Parameters:
*   edt_tbl:   contains all parameters and values to be modified for data
*              on source device
*   fileid:    coded portion of file name for display file
*
* Output Parameters:
*
* Return Values:
*   NULL_ERR = okay
*   negative = see [dspscr]
*
* Functions Referenced:
*   chgval:    change a screen value in the Presentation Area
*   dsperr:    display error/status message on Monitor
*   dspscr:    present menu/display on the Monitor
*   movcsr:    move the visible cursor on the Monitor
*   prmtch:    find parameter code match in parameter table
*   txtdec:    convert text string to decimal integer value
*   txthex:    convert text string to hexadecimal number
*   wgetch:    get a character from the keyboard through the window
*   wvalue:    save updated values in value file
*
* Global Variables Referenced:
*   input_line: Input Line window of display
*   rec_prm:    all recording parameters, lengths, types etc.
*   scr:        attributes and description of fields on current screen
*
* Description:
*   Presents the Replace A Parameter Value Display and accepts and validates
*   the keyboard entries and cursor motions made by the Operator to set up
*   the parameters and changes

```

```

*
* Algorithm:
*   begin
*       set return value to NULL_ERR
*       initialize replace value table
*       call [dspscr] to display menu
*       if error from [dspscr] then
*           return status value
*       set input response to 0
*       while input response is not ((CTRL Z and no errors) or CTRL C) do
*           begin
*               call [wgetch] to get keyboard input
*               call [movcsr] to move cursor if required
*               if status from [movcsr] is NOT_CRSR_STS or input is CTRL Z then
*                   begin
*                       call [chgval] to change the screen value
*                       if status from [chgval] is INP_LINE_STS or input is CTRL Z then
*                           begin
*                               set field layout table pointer (fltp) to "right" component
*                               cursor start
*                               set table pointer to 0
*                               while no errors and ptr less than end of rep_val table do
*                                   begin
*                                       call [prmtch] to verify existence of parameter
*                                       if error from [prmtch] then
*                                           call [dsperr] with "invalid parameter code"
*                                       else
*                                           begin
*                                               save [prmtch] pointer in a temporary table
*                                               move to next field at right via fltp
*                                               if parameter type is hexadecimal then
*                                                   call [txthex] to convert text to hexadecimal value
*                                               else
*                                                   call [txtdec] to convert text to decimal value
*                                               if no errors then
*                                                   begin
*                                                       save value in a temporary table
*                                                       move to next field at right via fltp
*                                                       increment table pointer
*                                                   end
*                                           end
*                                       end
*                                   end
*                               end
*                           end
*                       end
*                   end
*               if input response is CTRL Z then

```

2100-12-002

```
*      begin
*      call [wvalue] to save updated values
*      for each temporary table entry do
*          save in edit table
*      end
*      return status value
*  end
*
*****/
```


8.1.5 Replace a Digit in a Parameter Value

```

/*****
*
* Name:
*   rdigit
*
* Function:
*   Defines those parameter values for which digits are to be replaced
*
* Synopsis:
*   rdigit(fileid,edt_tbl)
*
* Input Parameters:
*   edt_tbl:   contains all parameters and values to be modified for data
*              on source device
*   fileid:    coded portion of file name for display file
*
* Output Parameters:
*
* Return Values:
*   NULL_ERR = okay
*   negative = see [dspscr]
*
* Functions Referenced:
*   chgval:    change a screen value in the Presentation Area
*   dsperr:    display error/status message on Monitor
*   dspscr:    present menu/display on the Monitor
*   movcsr:    move the visible cursor on the Monitor
*   prmtch:    find parameter code match in parameter table
*   txtdec:    convert text string to decimal integer value
*   wgetch:    get a character from the keyboard through the window
*   wvalue:    save updated values in value file
*
* Global Variables Referenced:
*   input_line: Input Line window of display
*   rec_prm:    all recording parameters, lengths, types etc.
*   scr:        attributes and description of fields on current screen
*
* Description:
*   Presents the Replace A Parameter Value Display and accepts and validates
*   the keyboard entries and cursor motions made by the Operator to set up
*   the parameters and changes
*

```

```

* Algorithm:
*   begin
*       set return value to NULL_ERR
*       initialize replace digit table
*       call [dspscr] to display menu
*       if error from [dspscr] then
*           return status value
*       set input response to 0
*       while input response is not ((CTRL Z and no errors) or CTRL C) do
*           begin
*               call [wgetch] to get keyboard input
*               call [movcsr] to move cursor if required
*               if status from [movcsr] is NOT_CRSR_STS or input is CTRL Z then
*                   begin
*                       call [chgval] to change the screen value
*                       if status from [chgval] is INF_LINE_STS or input is CTRL Z then
*                           begin
*                               set field layout table pointer (fltp) to "right" component
*                               cursor start
*                               set table index to 0
*                               while no errors and index less than end of rep_dig table do
*                                   begin
*                                       call [prmtch] to verify existence of parameter
*                                       if return status from [prmtch] not 0 then
*                                           call [dsperr] with "invalid parameter code"
*                                       else if parameter is NULL parameter then
*                                           move to next field down via fltp
*                                       else
*                                           begin
*                                               save [prmtch] pointer in a temporary table
*                                               move to next field at right via fltp
*                                               call [txtdec] to convert position to decimal value
*                                               if position is greater than length of parameter or
*                                                  is zero then
*                                                   call [dsperr] with "out of range"
*                                           else
*                                               begin
*                                                   save (position - 1) in a temporary table
*                                                   move to next field at right via fltp
*                                                   set valid sign flag accordingly
*                                                   set first flag accordingly
*                                                   set signed parameter flag accordingly
*                                                   set valid digit flag accordingly
*                                                   if not: valid digit and (not signed or (signed and
*                                                       not first)) or
*                                                       signed and valid sign and first then

```

2100-12-002

```
*          call [dsperr] with "invalid character"
*      else
*          begin
*              save digit in a temporary table
*              move to next field at right via fltp
*          end
*      end
*  end
*      increment table pointer
*  end
*      end
*  end
*      end
*  if input response is CTRL Z then
*      begin
*          call [wvalue] to save updated values
*          for each temporary table entry do
*              save in edit table
*          end
*      return status value
*  end
*
*****/
```

2100-12-002

8.1.6 Delete/Create a Parameter

```

/*****
*
* Name:
*   delprm
*
* Function:
*   Defines those parameter values to be deleted/added
*
* Synopsis:
*   delprm(fileid,edt_tbl)
*
* Input Parameters:
*   edt_tbl:   contains all parameters and values to be modified for data
*              on source device
*   fileid:    coded portion of file name for display file
*
* Output Parameters:
*
* Return Values:
*   NULL_ERR = okay
*   negative = see [dspscr]
*
* Functions Referenced:
*   chgval:    change a screen value in the Presentation Area
*   dspscr:    present menu/display on the Monitor
*   movcsr:    move the visible cursor on the Monitor
*   vfydap:    verify deletion/addition parameters
*   wgetch:    get a character from the keyboard through the window
*   wvalue:    save updated values in value file
*
* Global Variables Referenced:
*   input_line: Input Line window of display
*
* Description:
*   Presents the Delete/Add Parameters Display and accepts and validates
*   the keyboard entries and cursor motions made by the Operator to set up
*   the parameters and calculations
*
* Algorithm:
*   begin
*       set return value to NULL_ERR
*       initialize temporary delete and add parameters tables

```

2100-12-002

```
*      call [dspscr] to display menu
*      if error from [dspscr] then
*          return status value
*      set input response to 0
*      while input response is not ((CTRL Z and no errors) or CTRL C) do
*          begin
*              call [wgetch] to get keyboard input
*              call [movcsr] to move cursor if required
*              if status from [movcsr] is NOT_CRSR_STS or input is CTRL Z then
*                  begin
*                      call [chgval] to change the screen value
*                      if status from [chgval] is INP_LINE_STS or input is CTRL Z then
*                          call [vfydap] to verify the Operator's choices for
*                          parameter deletion/addition
*                  end
*              end
*          end
*      if input response is CTRL Z then
*          begin
*              call [wvalue] to save updated values
*              for each temporary table entry do
*                  save in edit table
*              end
*          end
*      return status value
*  end
*
*****/
```

8.1.6.1 Verify Deletion/Creation Parameters

```

/*****
*
* Name:
*   vfydap
*
* Function:
*   Verify parameters for batch, parameter deletion/addition
*
* Synopsis:
*   vfydap(del_prm,add_prm)
*
* Input Parameters:
*   del_prm:  parameters to be deleted during batch edits
*   add_prm:  parameters to be added during batch edits
*
* Output Parameters:
*
* Return Values:
*   NULL_ERR      = okay
*   INV_P_CODE_ERR = invalid recording parameter code
*
* Functions Referenced:
*   dsperr:  display message on Error Line window
*   prmtch:  match parameter against those of recording parameter table
*   prscal:  parse the calculation text to verify syntax
*
* Global Variables Referenced:
*   rec_prm:  table containing all recording parms, lengths, types etc.
*   scr:      attributes and descriptions of values on screen
*
* Description:
*   Verifies the parameters to be deleted and added, and parses the text
*   string defining the calculation for parameters to be added.
*
* Algorithm:
*   begin
*       set return value to NULL_ERR
*       set field layout table pointer (fltp) to field at right of Input Line
*       set table pointer to 0
*       while no errors and table pointer less than SML_SIZ_TBL do
*           begin
*               call [prmtch] to match field value against recording parameters

```

2100-12-002

```
*      if error from [prmtch] then
*          call [dsperr] to display message "invalid parameter code"
*      else
*          begin
*              save returned index in delete table
*              increment table pointer
*              increment fltp
*          end
*      end
*      set table pointer to 0
*      while no errors and table pointer less than SML_SIZ_TBL do
*          begin
*              call [prmtch] to check for parameter's existence
*              if error returned from [prmtch] then
*                  call [dsperr] to display message "invalid parameter code"
*              else
*                  begin
*                      move to next field (fltp)      { formula }
*                      if parameter not the NULL parameter then
*                          begin
*                              call [prscal] to parse calculation text
*                              if no errors from [prscal] then
*                                  save parameter code in add table
*                              end
*                          end
*                      move to next field (fltp)
*                      increment table pointer
*                  end
*              end
*          end
*      return status value
*  end
*
*****/
```

8.1.6.2 Parse Creation Parameter Formula

```

/*****
*
*   Name:
*       prscal
*
*   Function:
*       Parses calculation text
*
*   Synopsis:
*       prscal(fld_ptr,add_prm)
*
*   Input Parameters:
*       fld_ptr:    pointer to screen field
*       add_prm:    specifications for all added parameters
*
*   Output Parameters:
*
*   Return Values:
*       NULL_ERR      = okay
*       INV_CONST_ERR  = invalid constant specified in expression
*       INV_P_CODE_ERR = nonexistent parameter code
*       SYNTAX_ERR     = syntax error (see Design Document)
*       TBL_FULL_ERR   = too many operators or operands have been specified
*
*   Functions Referenced:
*       dsperr:  display error on Error Line window of screen
*       prmtdc:  verify parameter codes exists against recording parm table
*       txtdec:  convert text to decimal integer
*
*   Global Variables Referenced:
*       rec_prm: recording parameters, types, lengths etc
*       scr:     contains attributes and text on Monitor
*
*   Description:
*       Parses the calculation entered for a parameter and converts recording
*       parameter codes to table indices and text strings to constants
*
*   Algorithm:
*       begin
*           set return value to NULL_ERR
*           set operator and operand table indices to 0
*           set op flag to FALSE

```



```

*      set character pointer to start of text field
*      while no errors and not at end of field do
*          begin
*              skip spaces
*              copy text to a temporary string (until space encountered)
*              if number of characters copied is 1 then
*                  begin
*                      if character is '+', '-', '/', or '*' then
*                          begin
*                              if op flag is FALSE then
*                                  call [dsperr] to display message "syntax error"
*                                      ( operand must precede operator )
*                              else
*                                  begin
*                                      if operator table full then
*                                          call [dsperr] with "table full"
*                                              ( size of expression limited by number of operators )
*                                      else
*                                          begin
*                                              save operator in edit table
*                                              set op flag to FALSE
*                                          end
*                                      end
*                                  end
*                              else if operator flag is TRUE then
*                                  call [dsperr] with "invalid character"
*                                      ( invalid operator )
*                              else if character is not '0' ... '9' then
*                                  call [dsperr] to display message "invalid constant"
*                              else if operand table is full then
*                                  call [dsperr] with message "table full"
*                                      ( size of expression limited by number of operands )
*                              else
*                                  begin
*                                      save value in edit table
*                                      set tag field to indicate constant
*                                      set op flag to TRUE
*                                  end
*                              end
*                          else if number of characters is two then
*                              begin
*                                  if op flag is TRUE then
*                                      call [dsperr] to display message "syntax error"
*                                          ( operator must precede operand )
*                                  else if first char is a '-' or '0' ... '9' and
*                                      second char is a '0' .. '9' then

```

```

*           begin
*           save value in edit table
*           set tag field to indicate constant
*           set op flag to TRUE
*           end
*       else
*       begin
*       call [prmtch] to determine if parameter code exists
*       if return value from [prmtch] is error then
*           call [dsperr] to display message "invalid parameter code"
*           ( two characters are not a parameter code )
*       else
*       begin
*       set tag field to indicate index
*       save index in edit table
*       set op flag to TRUE
*       end
*       end
*       end
*       else if number of characters is not 0 (ie. field not empty) then
*       begin
*       if operator flag is TRUE then
*           call [dsperr] with "syntax error"
*           ( operator must precede operand )
*       else if operand table is full then
*           call [dsperr] with "table full"
*           ( expression size limited by number of operands )
*       else
*       begin
*       call [txtdec] to convert text to integer value
*       if no errors then
*       begin
*       set tag field to indicate constant
*       save value in edit table
*       set op flag to TRUE
*       end
*       end
*       end
*       end
*       if op flag is FALSE and no errors then
*       call [dsperr] to display message "syntax error"
*       ( expression must terminate with an operand )
*       return status value
*       end
*
*****/

```

2100-12-002

8.1.7 Define Spike Checking Specifications

```

/*****
*
* Name:
*     spkchk
*
* Function:
*     Defines spike-checking specifications
*
* Synopsis:
*     spkchk(fileid,edt_tbl)
*
* Input Parameters:
*     edt_tbl:  contains all parameters and values to be modified for data
*               on source device
*     fileid:   coded portion of file name for display file
*
* Output Parameters:
*
* Return Values:
*     NULL_ERR = okay
*     negative = see [dspscr]
*
* Functions Referenced:
*     chgval:   change a screen value in the Presentation Area
*     dsperr:   display error/status message on Monitor
*     dspscr:   present menu/display on the Monitor
*     movcsr:   move the visible cursor on the Monitor
*     prmtch:   find parameter code match in parameter table
*     txtdec:   convert text string to decimal integer value
*     wgetch:   get a character from the keyboard through the window
*     wvalue:   save updated values in value file
*
* Global Variables Referenced:
*     input_line:  Input Line window of display
*     rec_prm:     all recording parameters, lengths, types etc.
*     scr:         attributes and description of fields on current screen
*
* Description:
*     Presents the Spike Check Display and accepts and validates the keyboard
*     entries and cursor motions made by the Operator to set up the parameters
*     and changes
*
*****/

```

```

* Algorithm:
*   begin
*       set return value to NULL_ERR
*       initialize temporary spike check table
*       call [dspscr] to display menu
*       if error from [dspscr] then
*           return status value
*       set input response to 0
*       while input response is not ((CTRL Z and no errors) or CTRL C) do
*           begin
*               call [lwgetch] to get keyboard input
*               call [movcsr] to move cursor if required
*               if status from [movcsr] is NOT_CRSR_STS or input is CTRL Z then
*                   begin
*                       call [chgval] to change the screen value
*                       if status from [chgval] is INP_LINE_STS or input is CTRL Z then
*                           begin
*                               set field layout table pointer (fltp) to field to right of
*                                   Input Line
*                               set table pointer to 0
*                               while no errors and index less than end of spike table do
*                                   begin
*                                       call [prmtch] to verify existence of parameter
*                                       if return status from [prmtch] not 0 then
*                                           call [dsperr] with "invalid parameter code"
*                                       else if parameter is NULL parm then
*                                           move to next field down via fltp
*                                       else
*                                           begin
*                                               save [prmtch] pointer in a temporary table
*                                               move to next field at right via fltp
*                                               call [txtdec] to convert percentage change to decimal
*                                               if no error then
*                                                   begin
*                                                       save change in a temporary table
*                                                       move to next field at right via fltp
*                                                       call [txtdec] to convert range to decimal
*                                                       if no error then
*                                                           if range is greater or equal to maximum then
*                                                               call [dsperr] with "limit error"
*                                                           else
*                                                               begin
*                                                                   save range in a temporary table
*                                                                   move to next field at right via fltp
*                                                               end
*                                                           end
*                                                       end
*                               end
*                           end
*                       end
*                   end
*               end
*           end
*       end

```

2100-12-002

```
*           end
*           end
*           increment table pointer
*           end
*       end
*   end
*   if input response is CTRL Z then
*       begin
*           call [wvalue] to save updated values
*           for each temporary table entry do
*               save in edit table
*           end
*       return status value
*   end
*
*****/
```

8.1.8 Batch Editor Process

```

/*****
*
* Name:
*   edtchg
*
* Function:
*   Perform all batch editing changes
*
* Synopsis:
*   edtchg(edt_tbl)
*
* Input Parameters:
*   edt_tbl:  all parameters and values to be modified
*
* Output Parameters:
*
* Return Values:
*   NULL_ERR = okay
*   negative = see [ioopen]
*
* Functions Referenced:
*   asctime:  convert time value to ascii string
*   dectxt:   convert decimal integer to text
*   dsperr:   display message on Error Line of Monitor
*   fclose:   close file for access
*   fopen:    open a file for access
*   fprintf:  write to terminal
*   hextxt:   convert hex value to text
*   ioclos:   close device for access
*   ioopen:   open a device for access
*   ioread:   read a data block from a device
*   sserch:   search for a string in a data block
*   time:     get system time and date
*   txtdec:   convert text to decimal value
*   txthex:   convert text to hexadecimal value
*   wrtblk:   write a data block to a device
*
* Global Variables Referenced:
*   buffer1:   data block storage area
*   buffer2:   data block storage area
*   lun_table: maps logical unit numbers to physical devices
*   rec_prm:   recording parameters and specifications

```

```

*
* Description:
*   Performs all batch edit changes as defined in the edit table. This
*   continues until end of source device, end of destination device, or
*   an error is encountered.
*
* Algorithm:
*   begin
*       set return value to NULL_ERR
*       call [iopen] to open source device
*       if no error from [iopen] then
*           begin
*               call [iopen] to open destination device
*               if error from [iopen] then
*                   call [ioclos] to close source device
*               end
*           if error encountered then
*               return status value
*           set block counter to 0
*           set done flag to FALSE
*           set edit flag to FALSE
*           while not done and no errors do
*               begin
*                   call [ioread] to read a block from source device
*                   if error or end of source device then
*                       begin
*                           set done flag to TRUE
*                           if error then
*                               call [dsperr] with "error at source device"
*                           end
*                       else if first block then
*                           begin
*                               calculate actual size of data block
*                               set length of new block to length of current data block
*                               call [sserch] to find basis parameter in data block
*                               if error from [sserch] then
*                                   call [dsperr] with message "missing parameter code"
*                               else
*                                   begin
*                                       save the data block pointer for the basis parameter
*                                       call [sserch] again to find second occurrence of basis parm.
*                                       in data block and use the difference as record length
*                                       calculate length of header in data block
*                                   end
*                               while no errors and not all entries have been checked do
*                                   begin

```



```

*      if the "add/subtract to/from parm" table entry exists then
*      begin
*      call [sserch] to find parameter
*      if error from [sserch] then
*      call [dsperr] with message "missing parameter code"
*      else
*      save rec parm table index in edit table
*      end
*      if the "replace value" parm. table entry exists then
*      begin
*      call [sserch] to find parameter
*      if error from [sserch] then
*      call [dsperr] with message "missing parameter code"
*      else
*      save rec parm table index in edit table
*      end
*      if the "replace digit" parm. table entry exists then
*      begin
*      call [sserch] to find parameter
*      if error from [sserch] then
*      call [dsperr] with message "missing parameter code"
*      else
*      save rec parm table index in edit table
*      end
*      if the "spike check" parm. table entry exists then
*      begin
*      call [sserch] to find parameter
*      if error from [sserch] then
*      call [dsperr] with message "missing parameter code"
*      else
*      save rec parm table index in edit table
*      end
*      increment table pointer
*      end
*      while no errors and not all entries have been checked do
*      begin
*      if the "delete" parm. table entry exists then
*      begin
*      call [sserch] to find parameter
*      if error from [sserch] then
*      call [dsperr] with message "missing parameter code"
*      else
*      begin
*      save rec parm table index in edit table
*      subtract length of parameter from new block length
*      end

```

```

*           end
*       end
*   while no errors and not all entries have been checked do
*       begin
*           if the "insert" parm. table entry exists then
*               begin
*                   call [sserch] to find parameter
*                   if error from [sserch] then
*                       call [dsperr] with message "missing parameter code"
*                   else
*                       begin
*                           save rec parm table index in edit table
*                           add length of parameter to new block length
*                           while no errors and not at end of operand table do
*                               if operand is a parameter value then
*                                   begin
*                                       call [sserch] to find parameter in data block
*                                       save data block index for parameter
*                                   end
*                               end
*                           end
*                       end
*                   end
*               end
*           end
*       calculate length of new record
*       end
*   set record pointer to 0
*   while no errors and record pointer is less than 10 do
*       begin
*           if basis parameter type is hex then
*               call [txthex] to get basis parameter value
*           else
*               call [txtdec] to get basis parameter value
*           if no error from conversion then
*               if value is between the two limits then
*                   set edit flag to TRUE
*               else if block count is 0 and record count is 0 and
*                   value is beyond stop limit or start limit then
*                   call [dsperr] with "limit error"
*               else if edit flag is TRUE then
*                   set done flag to TRUE
*               while no errors and edit flag set and
*                   not all spike table entries examined do
*                   begin
*                       if parm code for spike-check table not NULL parm then
*                           begin
*                               if parameter type is hexadecimal then
*                                   call [txthex] to convert text to hex integer

```

```

*           else
*           call [txtdec] to covert text to decimal value
*       if no errors from conversion then
*       begin
*       save value in temporary spike value table (svt)
*       if enough values exist in table for range to be
*       verified against then
*       begin
*       get comparison value from svt
*       subtract comparison value from current value
*       if change is greater than that specified then
*       begin
*       call [time] to get current time
*       call [asctime] to convert to a string
*       call [fopen] to open error console
*       call [fprintf] with standard error device
*       "spike at block b, record r, value v,
*       change p"
*       call [close] to close the error device
*       end
*       end
*       end
*       while no errors and edit flag set and
*       not all change-one-digit entries examined do
*       begin
*       if parm index in change-one-digit table not 0 then
*       change digit in data block buffer
*       end
*       while no errors and edit flag set and
*       not all replace-value entries examined do
*       begin
*       if parm index in replace-value table not 0 then
*       begin
*       if parameter type is decimal then
*       call [dectxt] to convert replacement value from
*       decimal to text in temporary buffer
*       else
*       call [hextxt] to convert replacement value from
*       hex value to text in temporary buffer
*       end
*       while no errors and edit flag set and
*       not all add/subtract to/from parameter entries examined do
*       begin
*       if parm index in add-value-to-parm table not 0 then
*       begin
*       if parameter type is decimal then

```

```

*           call [txtdec] to convert text (old value) to decimal
*       else
*           call [txthex] to convert text (old value) to hex
*   if no errors then
*       begin
*           add new value to old value
*           if parameter type is decimal then
*               call [dectxt] to convert new decimal value to text
*           else
*               call [hextxt] to convert new hex value to text
*           save text in temporary buffer
*       end
*   end
* while no errors and edit flag set and
*     not all add-parameter table entries examined do
*   begin
*   if parm index in add-parameter table not 0 then
*       begin
*           set build value to 0
*           set operator flag to '+'
*           set quit flag to FALSE
*           while no errors and quit false is FALSE do
*               begin
*                   if tag field indicates a constant then
*                       get operand from the table
*                   else if parm. type is hexadecimal then
*                       call [txthex] to get operand from data block
*                   else
*                       call [txtdec] to get operand from data block
*                   case operator of
*                       '+': add value to build
*                       '-': subtract value from build
*                       '*': multiple build by value
*                       '/': if value is 0 then
*                           call [dsperr] with "divide by zero"
*                       else
*                           divide build by value
*                   default:
*                       call [dsperr] with "syntax error"
*                   if at end of operator table or no more operators then
*                       set quit flag to TRUE
*                   else
*                       get next operator from operator table
*                   end
*           if no errors then
*               begin

```

```

*           if parameter type is decimal then
*             call [dectxt] to append value to end of temp buffer
*           else
*             call [hextxt] to append hex value to temp buffer
*             append parameter code to end of temporary buffer
*           end
*         end
*       end
*     while no errors and edit flag set and
*       not all delete-parameter table entries examined do
*     begin
*       if parm index in delete-parm table not 0 and
*       begin
*         fill in parm code and value with spaces in data buffer
*       end
*     if no errors then
*     begin
*       for each character in record of data buffer do
*         if character is not space then
*           copy data buffer character to new block buffer
*         end
*       increment record pointer
*     end
*   if no errors then
*   begin
*     copy header to new data block
*     place the null character at the end of the new block
*     call [wrtblk] to write new data block
*   end
*   increment block counter
* end
* if no errors or end of source device has been encountered then
* begin
*   call [wrtblk] to complete dumping to output device
*   call [dsperr] with "batch editing complete"
* end
* call [close] to close source device
* call [close] to close destination device
* return status value
* end
*
*****/

```

8.2 Screen Editor

```

/*****
*
*   Name:
*       scredt
*
*   Function:
*       Full screen data block editor
*
*   Synopsis:
*       scredt(fileid,edt_tbl)
*
*   Input Parameters:
*       fileid:    coded portion of file name for display file
*       edt_tbl:   values, parameters, limits etc. for editing blocks
*
*   Output Parameters:
*
*   Return Values:
*       NULL_ERR   = okay
*       GENERAL_ERR = no lun specified for the IBM-AT Monitor
*       negative    = see [ioopen], [dspscr]
*
*   Functions Referenced:
*       addch:      add a character to the standard screen window
*       chkblk:     check airborne data block for errors
*       dsperr:     display status/error message on Error Window of monitor
*       dspscr:     present display and prompt on screen
*       echo:       turn on keyboard echoing of characters
*       getyx:      get coordinates of cursor
*       inpstr:     get data from keyboard
*       ioctl:      Venix device control routine
*       ioclos:     close a device for access
*       ioopen:     open a device for access
*       ioread:     read a data block from device
*       iowrit:     write a data block to device
*       lseek:      seek to logical address
*       move:       move visible cursor
*       noecho:     turn off keyboard echoing of characters
*       printf:     send output to the screen
*       refresh:    make changes on monitor as made to standard screen window
*       standend:   turn highlighting off
*       standout:   turn highlighting on

```

```

*      strlen:   finds length of string (terminated by null character)
*      wgetch:   get a character through the window
*      wrtblk:   write a block of data to a device
*
* Global Variables Referenced:
*      air_chk_set:  airborne data checking set and header
*      buffer1:     data block storage area #1
*      d_map0:      removable cartridge disk map unit # 0
*      d_map1:      removable cartridge disk map unit # 1
*      input_line:  Input Line window of display
*      lun_tbl:     table mapping logical unit numbers to physical devices
*      rec_prm:     table of recording parameters and their attributes
*
* Description:
*      Provides a full-screen editor to read, modify, and write blocks of
*      airborne or diurnal data.
*
* Algorithm:
*      begin
*          set return value to NULL_ERR
*          call [dspscr] to display editor screen (initially blank)
*          if error from [dspscr] then
*              return status value
*          set screen lun table pointer (sltp) to negative value
*          while sltp is negative and not at end of lun table do
*              if lun table entry is MTR then
*                  set sltp to lun table index
*          if sltp is still negative then
*              call [dsperr] with general error message
*          else
*              begin
*                  call [ioopen] to access source device
*                  if no error then
*                      begin
*                          call [ioopen] to access destination device
*                          if error from [ioopen] then
*                              call [ioclos] to terminate access of source device
*                          end
*                      end
*              end
*          if error then
*              return status value
*          set insert flag to FALSE
*          set input response to 0
*          set start of text to 0
*          set end of text to 0
*          set block length to maximum size

```

```

*       while input response not CTRL Z or CTRL C do
*       begin
*       call [wgetch] to get keyboard input
*       call [getyx] to determine cursor position
*       case input of
*       begin
*       CTRL Z:
*       call [echo] to turn on keyboard echoing
*       call [inpstr] to prompt with "ARE YOU SURE?"
*       call [noecho] to turn off keyboard echoing
*       if input response was 'y' or 'Y' then
*       call [wrtblk] to conclude write to destination device
*       CTRL C:
*       no operation
*       ENTER:
*       if insert mode then
*       begin
*       call [printf] to sound bell -- insert mode cancelled
*       set insert flag to FALSE
*       call [dsperr] to clear "insert mode" text
*       end
*       else
*       begin
*       call [move] to set cursor at input line window
*       call [refresh] to make change on the screen
*       end
*       ESCAPE:
*       call [wgetch] to get the second part of the keystroke
*       case second key of
*       begin
*       HOME:
*       call [move] to set cursor at top, left of present. area
*       call [refresh] to make change on the screen
*       RIGHT CURSOR:
*       if cursor is in the presentation area then
*       begin
*       set x and y coordinates to home position
*       end
*       else
*       begin
*       if at the end of the data block on screen then
*       call [printf] to sound bell -- cannot move
*       else if x coordinate is less than right
*       edge of presentation area then
*       increment x coordinate
*       else

```



```

*           begin
*           if y coordinate is not at bottom line then
*               begin
*                   set x coordinate to beginning of line
*                   increment y coordinate
*               end
*           else
*               begin
*                   if moving down a line is possible then
*                       begin
*                           add a line length to the text start pointer
*                           if text end less text start is less than a
*                               screen full of characters then
*                               call [iowrit] with text end less start
*                               characters
*                           else
*                               call [iowrit] with a screen full of chars
*                           end
*                       end
*                   else
*                       call [printf] to sound bell -- at very
*                           bottom of text, cannot cursor
*                           down any further
*                   end
*               end
*           end
*           call [move] to position cursor
*           call [refresh] to update visible screen
* LEFT CURSOR:
*           if cursor is in the presentation area then
*               begin
*                   if nothing on screen then
*                       set Y coordinate to home position
*                   else if screen not full then
*                       set Y coordinate to end of text
*                   else
*                       set Y coordinate to bottom right corner
*                   if nothing on screen the
*                       set X coordinate to home position
*                   else if screen not full then
*                       set X coordinate to end of text
*                   else
*                       set X coordinate to bottom right corner
*                   end
*               end
*           else
*               begin
*                   if x coordinate is greater then left

```

```

*                                     edge of presentation area then
*                                     decrement x coordinate
*     else
*         begin
*             if y coordinate is not top line then
*                 begin
*                     set x coordinate to end of line
*                     decrement y coordinate
*                 end
*             else
*                 begin
*                     if start of text pointer is greater than an
*                       entire line of characters then
*                         begin
*                             subtract a line length from text start ptr
*                             if text end less text start less than a
*                               screen full of characters then
*                                 call [iowrit] with text end less start
*                                   characters
*                             else
*                                 call [iowrit] with a screen full of chars
*                             end
*                         else
*                             call [printf] to sound bell - at block start
*                         end
*                     end
*                 end
*             end
*         call [move] to position cursor
*         call [refresh] to update visible screen
*     DOWN CURSOR:
*         if cursor is in the presentation area then
*             begin
*                 set x and y coordinates to home position
*             end
*         else
*             begin
*                 if at the last line of the data block on screen then
*                     call [printf] to sound bell -- cannot move down
*                       any further
*                 else if not at bottom of presentation area then
*                     increment y coordinate
*                 else
*                     begin
*                         add a line length to the text start pointer
*                         if text end less text start less than a screen
*                           full of characters then

```

```

*           call [iowrit] with text end - start characters
*       else
*           call [iowrit] with a screen full of chars
*       if moving down a row would cause the cursor
*           to be in a non data field then
*           set x coordinate to end of data block cursor
*           location
*       end
*   end
*   call [move] to position cursor
*   call [refresh] to update visible screen
UP CURSOR:
*   if cursor is in the presentation area then
*       begin
*           if nothing on screen then
*               set Y coordinate to home position
*           else if screen not full then
*               set Y coordinate to end of text
*           else
*               set Y coordinate to bottom right corner
*           if nothing on screen the
*               set X coordinate to home position
*           else if screen not full then
*               set X coordinate to end of text
*           else
*               set X coordinate to bottom right corner
*           end
*       else
*           begin
*               if y coordinate is not top of presentation area then
*                   decrement y coordinate
*               else if test start less than a line of chars then
*                   call [printf] to sound bell -- cannot move up
*                   any further
*               else
*                   begin
*                       subtract a line length from the text start pointer
*                       if text end less text start less than a screen
*                           full of characters then
*                               call [iowrit] with text end - start characters
*                           else
*                               call [iowrit] with a screen full of chars
*                           end
*                   end
*               end
*           call [move] to reposition cursor
*           call [refresh] to make change on the screen

```

```

*
* PAGE DOWN:
*   if insert flag is set then
*       begin
*           call [printf] to sound bell -- cannot page while
*               inserting
*           set insert flag to FALSE
*           call [dsperr] to clear error line
*       end
*   else if text end less start provides a whole screen then
*       begin
*           add a screen's worth of chars. to text start
*           call [dsperr] to display "working ..."
*           if a full screen to be displayed then
*               call [iowrit] with a full screen of characters
*           else
*               call [iowrit] with text end less start characters
*           call [dsperr] to clear error line
*       end
*   else
*       call [printf] to sound bell -- at bottom of data
*           block
*
* PAGE UP:
*   if insert flag is set then
*       begin
*           call [printf] to sound bell -- cannot page while
*               inserting
*           set insert flag to FALSE
*           call [dsperr] to clear error line
*       end
*   else if text start not zero then
*       begin
*           if more than screen's worth of chars from start then
*               subtract a screen's worth from text start
*           else
*               set text start pointer to 0
*           call [dsperr] to display "working ..."
*           if a full screen to be displayed then
*               call [iowrit] with a full screen of characters
*           else
*               call [iowrit] with text end less start characters
*           call [dsperr] to clear error line
*       end
*   else
*       call [printf] to sound bell -- at top of data block
*   end
*
* INSERT:

```

```

*           if insert flag is set then
*               begin
*                   clear flag
*                   call [dsperr] to clear error line
*                   end
*           else
*               begin
*                   set flag to TRUE
*                   call [dsperr] with "*** insert mode ***"
*                   end
*       DELETE:
*           call [getyx] to get current cursor location
*           if at end of block then
*               call [printf] to sound bell -- cannot delete
*           else
*               begin
*                   copy buffer1 contents for i to i - 1 up to current
*                       cursor location
*                   decrement text end pointer
*                   if a full screen to be displayed then
*                       call [iowrit] with a full screen of characters
*                   else
*                       call [iowrit] with text end less start characters
*                   call [move] to reposition cursor
*                   call [refresh] to update screen
*                   end
*           default:
*               call [printf] to sound bell -- invalid key
*           end
*       CTRL R (read a block of data):
*           begin
*               call [dsperr] with "working ..."
*               call [ioread] to get block from source device
*               if no errors then
*                   begin
*                       if and block length is still set at maximum then
*                           call [strlen] to find actual length of block
*                       mark end of buffer with a special character ('~')
*                       reset text start to 0
*                       set text end to block length
*                       if a full screen to be displayed then
*                           call [iowrit] with a full screen of characters
*                       else
*                           call [iowrit] with text end less start characters
*                       end
*                   end
*       CTRL W (write a block of data):

```

```

*      change end of block character back to 'null' char
*      call [wrtblk] to send block to destination device
*      change end of block character back to special character
*      break
*  CTRL B (back up a block on medium):
*      if source device is magnetic tape unit then
*          call [ioctl] to backup one block
*      else if source device is hard disk then
*          begin
*              determine disk map applicable
*              if backing up is not past start of partition then
*                  begin
*                      determine number of bytes to back up
*                      call [lseek] to back up in Venix
*                      subtract the number of blocks reversed from lad pointer
*                  end
*              end
*          break
*      case CTRL E:
*          call [chkblk] to check data block for errors
*              (airborne data assumed)
*          if return status shows no error then
*              call [dsperr] to clear error line
*          else
*              begin
*                  set start of text to position of first error character
*                  if not a full page then
*                      call [iowrit] to display portion of page
*                  else
*                      call [iowrit] to display full page
*                  call [standout] to highlight errors
*                  while not all characters have been highlighted do
*                      begin
*                          call [move] to move to character
*                          call [addch] to rewrite character
*                      end
*                  call [move] to reposition cursor at start of error
*                  call [refresh] to update screen
*                  call [standend] to disable highlighting
*              end
*          break
*      default:
*          if key is not a printable character then
*              call [printf] to sound bell -- not printable
*          else (key is a replacement or insertion character)
*              begin

```

```

*      if at input line then
*          call [printf] to sound bell -- no chars allowed at
*              input line
*      else if insert flag is not set then
*          begin
*              if at end of data block then
*                  call [printf] to sound bell -- cannot add character
*              else
*                  begin
*                      call [addch] to add character to window
*                      add character to data buffer
*                      if x coordinate is not right edge of presentation
*                          area then
*                              call [move] to move right one position
*                      else if not at bottom of present. area then
*                          call [move] to move down and to start of line
*                      else
*                          begin
*                              add a line of characters to text start
*                              move to start of current line
*                          end
*                      call [refresh] to update screen
*                  end
*              end
*          else
*              begin
*                  move all characters in data buffer right one position
*                      from current cursor position to end
*                  add character to data buffer
*                  if x coordinate is not end of a line then
*                      call [move] to move right a position
*                  else if not at bottom of presentation area then
*                      call [move] to move down and to start of line
*                  else
*                      begin
*                          add a line of characters to text start
*                          move to start of line
*                      end
*                  if a full screen to be displayed then
*                      call [iowrit] with a full screen starting from
*                          current location
*                  else
*                      call [iowrit] with text end less start characters
*                          starting from current cursor location
*                  call [refresh] to update screen
*              end

```

2100-12-002

```
*           end
*         end
*       end
*     call [ioclos] to close the source device
*     call [ioclos] to close the destination device
*     call [dsperr] to clear error line
*     return status value
*   end
*
*****/
```


9.0 TEXT/NUMERIC CONVERSION ROUTINES

9.1 Convert Ascii Text to Decimal Integer

```

/*****
*
* Name:
*     txtdec
*
* Function:
*     Convert text string to decimal integer
*
* Synopsis:
*     txtdec(buffer,count,fld,value)
*
* Input Parameters:
*     buffer:  character buffer of source text to be converted
*     fld:     field containing any errors of conversion
*     count:   number of characters to be converted
*
* Output Parameters:
*     value:   signed, long integer of converted text
*
* Return Values:
*     NULL_ERR      = okay
*     TOO_LONG_ERR  = text too long for conversion
*     INV_CHAR_ERR  = invalid character
*     INV_DATA_ERR  = invalid data
*
* Functions Referenced:
*     dsperr:  display message on Error Line window
*
* Global Variables Referenced:
*
* Description:
*     Converts a text string of specified length to a signed, long integer
*
* Algorithm:
*     begin
*         set return value to NULL_ERR
*         if count is greater than NBR_LEN then
*             call [dsperr] to display message "text too long for conversion"
*         else

```

```

*      begin
*      set build to 0
*      set character counter to 0
*      set space counter to 0
*      while character counter is less than count and no errors do
*      begin
*      get character from buffer
*      if character counter = 0 then
*      if character is '-' or 'w' or 's' then
*      begin
*      set sign flag to negative
*      increment character pointer
*      get next character
*      end
*      else
*      begin
*      set sign flag to positive
*      if character is '+' or 'e' or 'n' then
*      begin
*      increment character pointer
*      get next character
*      end
*      end
*      if character is a space then
*      begin
*      increment space counter
*      increment character pointer
*      end
*      else if character is not '0' .. '9' then
*      call [dsperr] to display message "invalid character"
*      else if space counter is not zero then
*      call [dsperr] with message "invalid data"
*      else
*      begin
*      set build to build * 10 + character - '0'
*      get next character from buffer
*      end
*      end
*      end
*      if no errors then
*      move build * sign to 'value'
*      return status value
*      end
*
*****/

```

9.2 Convert Ascii Text to Hexadecimal Integer

```

/*****
*
* Name:
*   txthex
*
* Function:
*   Convert text string to hexadecimal value (integer)
*
* Synopsis:
*   txthex(buffer,count,fld,value)
*
* Input Parameters:
*   buffer:  character buffer of source text to be converted
*   fld:     field containing any errors of conversion
*   count:   number of characters to be converted
*
* Output Parameters:
*   value:   signed, long integer of converted text
*
* Return Values:
*   NULL_ERR      = okay
*   TOO_LONG_ERR  = text too long for conversion
*   INV_CHAR_ERR  = invalid character
*
* Functions Referenced:
*   dsperr:  display message on Error Line window
*
* Global Variables Referenced:
*
* Description:
*   Converts a text string of specified length to a signed, long integer
*   which represents a hexadecimal number
*
* Algorithm:
*   begin
*       set return value to NULL_ERR
*       if count is greater than NBR_LEN then
*           call [dsperr] to display message "text too long for conversion"
*       else
*           begin
*               set build to 0
*               set character counter to 0

```

```

*      if first character is 's' or 'w' or '-' then
*      begin
*      set sign flag to -1
*      increment character counter
*      end
*      else
*      set sign to 1
*      if first character is 'n' or 'e' then
*      increment character counter
*      while character counter is less than count and no errors do
*      begin
*      if character is not '0' .. '9' or
*      'a' .. 'f' or 'A' .. 'F' then
*      call [dsperr] to display message "invalid character"
*      else
*      begin
*      if character is 'a' .. 'f' then
*      set digit to character - 'W'
*      else if character is 'A' .. 'F' then
*      set digit to character - '7'
*      else
*      set digit to character - '0'
*      set build to build * 16 + digit
*      increment character counter
*      end
*      end
*      end
*      if no errors then
*      move build * sign to 'value'
*      return status value
*      end
*
*****/

```

9.3 Convert Decimal Integer to Ascii Text

```

/*****
*
* Name:
*   dectxt
*
* Function:
*   Convert decimal integer to text string
*
* Synopsis:
*   dectxt(buffer,count,fld,pad,sign,scale,value)
*
* Input Parameters:
*   count:   maximum number of characters of conversion
*   fld:     pointer to field containing the error
*   pad:     padding character
*   sign:    type of sign required after conversion
*   scale:   number of decimal places
*   value:   signed, long integer of value to be converted
*
* Output Parameters:
*   buffer:  character buffer of converted text
*
* Return Values:
*   NULL_ERR      = okay
*   NO_ROOM_IN_BFR_ERR = no room in buffer for complete conversion
*   INV_SIGN_ERR   = invalid sign for value
*
* Functions Referenced:
*   dsperr:  display message on Error Line window
*   strncpy: copy n characters from string 1 to string 2
*
* Global Variables Referenced:
*
* Description:
*   Converts a long signed, integer representing a decimal number to a text
*   string of specified length
*
* Algorithm:
*   begin
*       set return value to NULL_ERR
*       move value to build
*       initialize temporary buffer to pad character

```

```

*      set character pointer to count - 1
*      if build is less than 0 then
*          begin
*              negate build (ie make positive)
*              case sign of
*                  SPACE (no sign) :
*                      '+' :
*                          move '-' to buffer as sign
*                          break
*                  'w' (west) :
*                      move 'e' (east) to buffer as sign
*                      break
*                  'n' (north) :
*                      move 's' (south) to buffer as sign
*                      break
*                  default :
*                      call [dsperr] with "invalid sign"
*                      break
*              negate build
*          end
*      else if sign is not SPACE then
*          move sign to temporary buffer
*      if build is 0 then
*          set first element of buffer to '0'
*      else
*          while no errors and chr ptr is greater than or equal to 0 and
*              build is greater than 0 do
*              begin
*                  set character = build mod 10 + '0'
*                  save character in buffer
*                  decrement character pointer
*                  set build = build / 10 (integer division)
*                  if build is greater than 0 and (character pointer < 0 or
*                      (character pointer = 0 and value was negative)) then
*                      call [dsperr] with message "not enough buffer space allocated"
*                  end
*      if no errors then
*          begin
*              if scale is negative then
*                  begin
*                      if first character position of buffer is free then
*                          begin
*                              for number of characters to left of decimal point do
*                                  move characters left one position in buffer
*                              place decimal place at scale position
*                          end

```

2100-12-002

```
*      else
*      begin
*      for number of scaled decimal characters do
*      move characters right one position in buffer
*      place decimal place at scale position
*      end
*      end
*      call [strncpy] copy temporary buffer to parameter buffer
*      end
*      return status value
*      end
*
*****/
```

9.4 Convert Hexadecimal Integer to Ascii Text

```

/*****
*
*   Name:
*       hextxt
*
*   Function:
*       Convert hexadecimal value to text string
*
*   Synopsis:
*       hextxt(buffer,count,fld,sign,scale,value)
*
*   Input Parameters:
*       count:   maximum number of characters of conversion
*       fld:     screen field containing any errors of conversion
*       sign:    sign character for value
*       scale:   number of decimal places
*       value:   signed, long integer of value to be converted
*
*   Output Parameters:
*       buffer:  character buffer of converted text
*
*   Return Values:
*       NULL_ERR      = okay
*       NO_ROOM_IN_BFR_ERR = no room in buffer for complete conversion
*       INV_SIGN_ERR   = invalid sign
*
*   Functions Referenced:
*       dsperr:  display message on Error Line window
*       strncpy: copy n characters from string 2 to string 1
*
*   Global Variables Referenced:
*
*   Description:
*       Converts a long signed, integer (representing a hexadecimal number)
*       to a text string of specified length
*
*   Algorithm:
*       begin
*           set return value to NULL_ERR
*           move value to build
*           initialize temporary buffer to all '0'
*           set character pointer to count - 1

```



```

*      if build is less than 0 then
*      begin
*      negate build (ie. make positive)
*      case sign of
*      begin
*      SPACE (no sign):
*      '+':
*          move '-' to buffer as sign
*          break
*      'w':
*          move 'e' (east) to buffer as sign
*          break;
*      'n':
*          move 's' (south) to buffer as sign
*          break;
*      default:
*          call [dsperr] with "invalid sign"
*          break;
*      end
*      end
*      else if sign is not SPACE then
*      move sign to temporary buffer
*      if build is 0 then
*      set first element of buffer to '0'
*      else
*      while no errors and chr ptr is greater than or equal to 0 do
*      begin
*      set character = build mod 16
*      if character is less than or equal to 9 then
*      add '0' to character to make numeric character
*      else
*      add '7' to character to make alpha. character
*      save character in buffer
*      decrement character pointer
*      set build = build / 16 (integer division)
*      if build is greater than 0 and character pointer = 0 then
*      call [dsperr] with message "not enough buffer space allocated"
*      end
*      if no errors then
*      begin
*      if scale is negative then
*      begin
*      if first character position of buffer is free then
*      begin
*      for number of characters to left of decimal point do
*      move characters left one position in buffer

```

2100-12-002

```
*           place decimal place at scale position
*           end
*       else
*           begin
*               for number of scaled decimal characters do
*                   move characters right one position in buffer
*                   place decimal place at scale position
*               end
*           end
*       call [strncpy] to copy temporary buffer to parameter buffer
*       end
*   return status value
* end
*
*/
```

APPENDIX A

FIELD CHECKING SYSTEM CONSTANTS

	page
A.0 SCOPE	1
A.1 BASE CONSTANTS	2
A.2 DISPLAY CONSTANTS	5
A.3 KEY CONSTANTS	7
A.4 DISK CONSTANTS	9
A.5 BLOCK PRINTING CONSTANTS	10
A.6 PLOTTING CONSTANTS	11
A.7 STACKED PROFILE CONSTANTS	12
A.8 GRADIENT PARAMETER CREATION CONSTANTS	13
A.9 EDITING CONSTANTS	14



#2100-12-002.01.0

A.0 SCOPE

This Appendix lists all the constants used within the Field Checking System. Each section describes the constants of one of the specific processes of the System, with the exception of the first four sections which detail constants common to all software routines.



A.1 BASE CONSTANTS

CONSTANT	VALUE	DESCRIPTION
X_IN_LINE	= 0	: start of input line X coordinate
Y_IN_LINE	= 24	: start of input line Y coordinate
X_ERR_LINE	= 0	: start of error line X coordinate
Y_ERR_LINE	= 22	: start of error line Y coordinate
X_PMT_LINE	= 0	: start of prompt line X coordinate
Y_PMT_LINE	= 23	: start of prompt line Y coordinate
X_PRS_AREA	= 0	: start of presentation area X coordinate
Y_PRS_AREA	= 0	: start of presentation area Y coordinate
END_PRS_AREA	= 21	: last row of presentation area
PMT_TXT_LEN	= 400	: length of prompt table text
ERR_TXT_LEN	= 1800	: length of error table text
NULL_PTR	= -1	: null pointer
NBR_LEN	= 8	: maximum number of digits for any parameter
LAST_LUN	= 57	: largest logical unit number
MSG_TXT_SIZE	= 1400	: length of text in a table
MSG_CNT	= 65	: maximum number of messages in a table
MNEM_LEN	= 5	: length, in chars, of rec. parm mnemonics
UNIT_LEN	= 5	: length, in chars, of rec. parm units
REC_TBL_LEN	= 90	: number of rec parm definitions available
RPAG_SIZ	= 30	: number of rec parms. per page
CHAR_SET_LEN	= 132	: length of character-checking set
MAX_FLD	= 150	: maximum number of fields on the screen
DSP_TXT_MAX	= 1600	: maximum number of display text chars
V_LINE_LEN	= 70	: length, in chars, of a Versatec line
M_LINE_LEN	= 76	: length, in chars, of IBM-AT Monitor line
BLK_LEN_MAX	= 5000	: maximum length of recording block
PRT_LIN_LEN	= 132	: char length of versatec raster print line
PLT_LIN_LEN	= 132	: char length of versatec raster plot line
MAX_SCT	= 132	: maximum number of sectors for a plot line
L_PER_CHR	= 20	: raster lines per character
BTS_CHR	= 16	: number of plot bits per character
GRAD_DEF_CNT	= 12	: number of definable gradient parameters
MED_SIZ_TBL	= 10	: length of a medium-sized table
SML_SIZ_TBL	= 5	: length of a small-sized table
OPER_TBL_SIZ	= 8	: max. # of operators per calculation
ERR_TBL_OSET	= 40	: offset for negative Error Table indices
PMT_TBL_OSET	= 7	: offset for negative Prompt Table indices
SML_WIN	= 1	: small window height
LRG_WIN	= 22	: large window height
DNAM_LEN	= 15	: length of device name string
LUN_TBL_LEN	= 9	: length of logical unit number table
NUL	= 0	: no device
MT0	= 1	: magnetic tape unit #0



```

MT1      =      2 : magnetic tape unit #1
HD0      =      3 : bernoulli disk unit #0
HD1      =      4 : bernoulli disk unit #1
VPR      =      5 : versatec printer
SPR      =      6 : star printer
VPP      =      7 : versatec printer/plotter (simultaneous)
MNR      =      8 : IBM-AT Monitor

```

```
flp(A,B,C,D,E,F,G,H,I) A=B=C=D=E=F=G=H=I=0;
```

```
A="/dev/null\0\0\0\0\0";\  
B="/dev/mtr0\0\0\0\0\0";\  
C="/dev/mtr1\0\0\0\0\0";\  
D="/dev/hd0\0\0\0\0\0\0\0";\  
E="/dev/hd1\0\0\0\0\0\0\0";\  
F="/dev/lp2\0\0\0\0\0\0\0";\  
G="/dev/lp\0\0\0\0\0\0\0\0";\  
H="/dev/spq2\0\0\0\0\0\0\0";\  
I="\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0".
```

#2100-12-002.01.0

```
HI_LAD  = 0x145ff : upper bound on user defined logical addrs
LO_IRG  = 00001 : lower bound on user defined interrecord gap
HI_IRG  = 32767 : upper bound on user defined interrecord gap
LO_ILV  = 0 : lower bound on user defined disk interleave
HI_ILV  = 32 : upper bound on user defined disk interleave
LO_TAV  = 1 : lower bound on user defined available tape
HI_TAV  = 32767 : upper bound on user defined available tape
LO_DNS  = 1 : lower bound on user defined rec. density
HI_DNS  = 32767 : upper bound on user defined rec. density
GAP_SCL = 1000 : scaling factor on user defined intrcrd gap
RD_MD   = 0 : open device for read mode
WT_MD   = 1 : open device for write mode
RW_MD   = 2 : open device for read/write mode
X_HOME  = 1 : screen x coordinate of home position
Y_HOME  = 1 : screen y coordinate of home position
WN_WD   = 78 : width of presentation area in columns
WN_HT   = 20 : height of presentation area in rows
MX_CR=WN_WD * WN_HT: maximum no. of chars in presentation area
TPLT_LIN_LEN= 264 : plot line length
PCO     = 3 : plot coordinate offset on spec. display
TBTS_CHR = 8 : number of bits per character field of plot
TCHR_SCT = 2 : number of chars. per sector of plot line
```



A.2 DISPLAY CONSTANTS

CONSTANT	VALUE	DESCRIPTION
FLEN	= 28 :	length of filename in characters
DSPFIL	= "/usr/emr/display/dsp" :	display filename prefix
TPL	= ".tpl" :	display template filename suffix
VAL	= ".val" :	display value filename suffix
PRMPFL	= "/usr/emr/display/prompt.txt" :	prompt file
ERORFL	= "/usr/emr/display/errors.txt" :	error text file
NBR_DIG	= 3 :	number of digits in filename
IDX_LEN	= 3 :	no. of digits in screen coordinate value
TBL_CODE	= 3 :	number of digits in text table index code
LUN_SPC_DID	= 110 :	logical unit number specs. fileid
REC_SPC_DID	= 120 :	recording parameter specs. fileid
AIR_SPC_DID	= 130 :	airborne char. set specs. fileid
DIU_SPC_DID	= 140 :	diurnal char. set specs. fileid
SDV_SPC_DID	= 150 :	storage device specs. fileid
MX_PG	= 3 :	number of recording parameter pages
SAVE_CALL	= "cp /usr/emr/display/dspl2*.val /usr/emr/tmp/dspl2*.sav &" :	backup rec. spec. value files
SVS_OSET	= 25 :	offset to src page number of copy call
SVD_OSET	= 49 :	offset to dst page number of copy call
RM_CALL	= "rm /usr/emr/tmp/dspl2*.sav &" :	remove backup value files
RMS_OSET	= 21 :	offset to src page number of restore call
RSTR_CALL	= "mv /usr/emr/tmp/dspl2*.sav /usr/emr/display/dspl2*.val &" :	restore rec. spec. value files
RSS_OSET	= 21 :	offset to src page number of restore call
RSD_OSET	= 49 :	offset to dst page number of restore call
CLR_STR	= ". ."	



#2100-12-002.01.0

: string used to clear Input Line window



A.3 KEY CONSTANTS

CONSTANT	VALUE	DESCRIPTION
NULL_KY	= 0 :	ASCII null character
CTRL_B_KY	= 2 :	ASCII control B
CTRL_C_KY	= 3 :	ASCII control C
CTRL_D_KY	= 4 :	ASCII control D
CTRL_E_KY	= 5 :	ASCII control E
BELL_KY	= 7 :	ASCII bell
CTRL_H_KY	= 8 :	ASCII control H
B_SPC_KY	= 8 :	ASCII backspace
TAB_KY	= 9 :	ASCII tab forward
CTRL_I_KY	= 9 :	ASCII control I
ENTER_KY	= 10 :	ASCII linefeed
FF_KY	= 12 :	ASCII formfeed
CRT_KY	= 13 :	ASCII carriage return
BACK_KY	= 15 :	ASCII backup
CTRL_P_KY	= 16 :	ASCII control P
CTRL_R_KY	= 18 :	ASCII control R
CTRL_W_KY	= 23 :	ASCII control W
CTRL_Z_KY	= 26 :	ASCII control Z
ESC_KY	= 27 :	ASCII escape
SPACE_KY	= 32 :	ASCII space
D_QUOTE_KY	= 34 :	ASCII double quote
S_QUOTE_KY	= 39 :	ASCII single quote
ASTRK_KY	= 42 :	ASCII '*'
PLUS_KY	= 43 :	ASCII plus sign
MINUS_KY	= 45 :	ASCII minus sign
SLASH_KY	= 47 :	ASCII '/'
ZERO_KY	= 48 :	ASCII '0'
NINE_KY	= 57 :	ASCII '9'
OSB_KY	= 91 :	ASCII '[' -- Open Square Bracket
CSB_KY	= 93 :	ASCII ']' -- Close Square Bracket
E_KY	= 101 :	ASCII 'e'
N_KY	= 110 :	ASCII 'n'
S_KY	= 115 :	ASCII 's'
W_KY	= 119 :	ASCII 'w'
BAR_KY	= 124 :	ASCII ' '
WAVE_KY	= 126 :	ASCII '~'

Escape keys: one keystroke, two keyboard reads

HOME_KY	= 71 :	home
UP_KY	= 72 :	cursor up key
PG_UP_KY	= 73 :	page up key



#2100-12-002.01.0

RIGHT_KY	=	77	:	cursor right key
LEFT_KY	=	75	:	cursor left key
DOWN_KY	=	80	:	cursor down key
PG_DN_KY	=	81	:	page down key
INS_KY	=	82	:	insert
DEL_KY	=	83	:	delete



A.4 DISK CONSTANTS

CONSTANT	VALUE	DESCRIPTION
OFF_LINE	= 0x00	: format disk off line
ON_LINE	= 0x80	: format disk on line
CMD_BYT1	= 0x00	: command byte #1 for format command
LAD_LEN	= 5	: number of bytes making up a logical address
LNS_PER_PG	= 10	: number of directory lines per page



A.5 BLOCK PRINTING CONSTANTS

CONSTANT	VALUE	DESCRIPTION
BID_LEN	= 70	: block id length in characters
HDR_LEN	= 264	: block header length in characters
LNBR_PTR	= 7	: pointer in data block to line number data
LNBR_LEN	= 8	: line number data length in digits
LNBR_OSET	= 49	: position in blk id of line number placement
DDT_PTR	= 0	: pointer in data block to date (day) data
DDT_LEN	= 3	: date (day) data length in digits
DDT_OSET	= 36	: position in blk id of date (day) placement
YDT_PTR	= 3	: pointer in data block to date (year) data
YDT_LEN	= 2	: date (year) data length in digits
YDT_OSET	= 46	: position in blk id of date (year) placement
BNBR_LEN	= 5	: block number length in digits
BNBR_OSET	= 10	: position in blk id of block number placement
BLN_LEN	= 4	: date (year) data length in digits
BLN_OSET	= 25	: position in blk id of date (year) placement
HDR_DAT_LEN	= 17	: length of header in an airborne data block
PAG_LEN	= 64	: number of lines on a print page
AIR_FMT	= 1	: airborne data print format
DIU_FMT	= 2	: diurnal data print format
NON_FMT	= 3	: unformatted data print format
AIR_PRT_ID	=	"BLOCK NO. ##### LENGTH ##### DAY ### YEAR ## LINE #####" : airborne data block print header
DIU_PRT_ID	=	"BLOCK NO. ##### LENGTH ##### DAY ### YEAR ##" : diurnal data block print header
UNF_PRT_ID	=	"BLOCK NO. ##### LENGTH #####"
BAD	=	": Bad Block (#####) encountered:" : unformatted data block print header
TM_OST	= 24	: number of characters in time string
BN_OST	= 39	: offset to block number in Bad Block message
ER_OST	= 4	: error message offset to console
LAD_MSG	=	" LAD = " : disk address of block containing error
LAD_LEN	= 5	: length of logical address in characters
HDR_MSG1	=	"Block Header = ["
HDR_MSG2	=	"]..." : header of block containing error (for error console)



A.6 PLOTTING CONSTANTS

CONSTANT	VALUE	DESCRIPTION
MAX_PLT_PRM	= 36 :	max. # of parms that may be plotted at once
DDT_OSET	= 4 :	offset to "day" component of header
YDT_OSET	= 14 :	offset to "day" component of header
LIN_OSET	= 24 :	offset to "line" component of header
VSCL_OSET	= 17 :	offset to "vert. scale" component of header
VSCL_LEN	= 2 :	number of characters forming vertical scale
INTV_OSET	= 42 :	offset to "fid. parm. interval comp. of hdr
INTV_LEN	= 7 :	number of characters forming fid. parm. intv
SCALE_LEN	= 7 :	number of characters forming a plot scale
F_UNIT_OSET	= 50 :	offset to "fid. parm." interval units
PMO	= 3 :	parameter offset in specifications display
PLT_HDR1_TPLT=	"DAY 000 YEAR 00 LINE 00000000 "	
	:	first line of header
PLT_HDR2_TPLT=	"VERTICAL SCALE = 00, FIDUCIAL INTERVAL = 0000000 UNITS "	
	:	second line of header
SUC_DIF_ID	= " *** SUCCESSIVE DIFFERENCE PLOT"	
	:	addition to header for succ. diff plots
PRT_IDX	= 0 * PRT_LIN_LEN	
	:	index into buffer2 of print line text
BNK_IDX	= 1 * PRT_LIN_LEN	
	:	index into buffer2 of blank line text
MNM_IDX	= 2 * PRT_LIN_LEN	
	:	index into buffer2 of mnemonic line text
SCL_IDX	= 3 * PRT_LIN_LEN	
	:	index into buffer2 of scale line text
UNT_IDX	= 4 * PRT_LIN_LEN	
	:	index into buffer2 of unit line text
SCT_IDX	= 5 * PRT_LIN_LEN	
	:	index into buffer2 of sector line text
HD1_IDX	= 6 * PRT_LIN_LEN	
	:	index into buffer2 of header #1 line text
HD2_IDX	= 7 * PRT_LIN_LEN	
	:	index into buffer2 of header #2 line text
PRE_LNS	= 5 :	lines to output prior to printing plot info.



A.7 STACKED PROFILE CONSTANTS

CONSTANT	VALUE	DESCRIPTION
STK_SIZ	= 757 :	size of stack table
MX_F_LN	= 24 :	maximum number of flights lines for profile
LEN_LIN_FIL	= 30 :	length of line file name
NAM_LIN_FIL	= "/usr/emr/tmp/ln"	
	:	first portion on line file name
DDT_OSET	= 19 :	offset to date-day in header
DDT_LEN	= 3 :	number of chars forming date-day
YDT_OSET	= 29 :	offset to date-year in header
YDT_LEN	= 2 :	number of chars forming date-year
F_TYP_OSET	= 27 :	offset to format type in header
F_UNT_OSET	= 62 :	offset to fmt. parm. units in header
P_PRM_OSET	= 20 :	offset to profile parameter in header
P_UNT_OSET	= 45 :	offset to prf. parm. units in header
PSCL_OSET	= 37 :	offset to profile parameter scale in header
FIN_T_OSET	= 54 :	offset to format interval in header
VSCL_OSET	= 17 :	offset to vertical scale in header
SEPR_OSET	= 42 :	offset to separation in header
SCL_LEN	= 7 :	number of chars forming scale
INT_LEN	= 7 :	number of chars forming interval
STK_HDR1	= "STACKED PROFILE	FORMAT:
	:	INTERVAL = ##### UNITS"
	:	first line of header
STK_HDR2	= "FLIGHT DATE: DAY ### YEAR ##"	
	:	second line of header
STK_HDR3	= "PROFILE PARAMETER =	SCALE =
	:	##### UNITS"
	:	third line of header
STK_HDR4	= "VERTICAL SCALE = ##,	SEPARATION =
	:	### SECTORS"
	:	fourth line of header
FREE	= "FREE"	
	:	free-format text
TIME_ADJ	= "TIME-ADJUSTED"	
	:	time-adjusted text
MAX_FLINE	= 30 :	max number of lines stacked per page
STK_DAT_FIL	= "/usr/emr/tmp/stkdat"	
	:	stack table data structure file
RM_FLNS	= "rm /usr/emr/tmp"	
	:	remove stacked flight line data from AT Disk



A.8 GRADIENT PARAMETER CREATION CONSTANTS

CONSTANT	VALUE	DESCRIPTION
PR1	= "KG" :	pair gradient parameter code #1
PR2	= "KH" :	pair gradient parameter code #2
PR3	= "KI" :	pair gradient parameter code #3
PR4	= "KJ" :	pair gradient parameter code #4
PR5	= "KK" :	pair gradient parameter code #5
PR6	= "KL" :	pair gradient parameter code #6
QD1	= "KM" :	quad gradient parameter code #1
QD2	= "KN" :	quad gradient parameter code #2
QD3	= "KO" :	quad gradient parameter code #3

PAIR = {PR1,PR2,PR3,PR4,PR5,PR6}
 : pair table

QUAD = {QD1,QD2,QD3}
 : quad table



#2100-12-002.01.0

A.9 EDITING CONSTANTS

CONSTANT	VALUE	DESCRIPTION
SPIKE_TBL_SIZ	= 100	: maximum number of spike table entries
OPER_TBL_SIZ	= 8	: max. # of operators per calculation



#2100-12-002.01.0

APPENDIX B

FIELD CHECKING SYSTEM PROMPTS

B.0	SCOPE	page 1
B.1	PROMPTS	2



#2100-12-002.01.0

B.0 SCOPE

This Appendix lists the prompt messages used within the Field Checking System. For each prompt, the internal constant, internal value and description/text will be given.



#2100-12-002.01.0

B.1 PROMPTS

CONSTANT	NUMBER	TEXT/DESCRIPTION
NULL_PMT	= 0 :	
ENTER_OPT_PMT	= -1 :	"enter number of option"
USE_KEYS_PMT	= -2 :	"use cursor keys and keyboard to set up specifications"
NBR_BLKs_PMT	= -3 :	"enter number of blocks"
STRNG_PMT	= -4 :	"enter string"
EDIT_PMT	= -5 :	screen editor help instructions
SURE_PMT	= -6 :	"are you sure?"
DMAP_PMT	= -7 :	disk directory display commands
RM_FILs_PMT	= -8 :	"remove stacked line data from AT Disk?"



#2100-12-002.01.0

APPENDIX C

FIELD CHECKING SYSTEM STATUSES/ERRORS

C.0	SCOPE	page 1
C.1	STATUSES	2
C.2	ERRORS	3



#2100-12-002.01.0

C.0 SCOPE

This Appendix lists the status and error messages used within the Field Checking System. Section C.1 describes the statuses. Section C.2 shows the error messages. For each status and error value, the internal constant and description/text will be given.

NOTE: some statuses do not have a text string associated with them: the value shown is used internally by the Software. These statuses will be so marked by an asterisk (*).



C.1 STATUSES

CONSTANT	NUMBER	TEXT/DESCRIPTION
NO_VALUE_FILE_STS	= *1	: no value file found
WORKING_STS	= 1	: working ...
NOT_CRSR_STS	= *2	: no cursor movement
INP_LINE_STS	= *3	: at Input Line window
STRT_BLK_PRT_STS	= 4	: block printing commencing ...
DONE_BLK_PRT_STS	= 5	: ... block printing complete
STRT_PLT_STS	= 6	: plotting commencing ...
DONE_PLT_STS	= 7	: ... plotting complete
STRT_SRT_STK_STS	= 8	: extraction commencing ...
DONE_SRT_STK_STS	= 9	: ... extraction complete
STRT_PRF_STK_STS	= 10	: stacked profiling commencing ...
DONE_PRF_STK_STS	= 11	: ... stacked profiling complete
STRT_GRD_CRT_STS	= 12	: gradient parm creation commencing ...
DONE_GRD_CRT_STS	= 13	: ... gradient parm creation complete
STRT_B_EDIT_STS	= 14	: batch edit commencing ...
DONE_B_EDIT_STS	= 15	: ... batch edit complete
END_SRC_DEV_STS	= 16	: end of source device found
INSRT_STS	= 17	: insert mode
CHNG_TAPE_STS	= 18	: change tape and hit f1 to resume
STRT_CPY_STS	= 19	: copy commencing ...
DONE_CPY_STS	= 20	: ... copy complete



C.2 ERRORS

CONSTANT	NUMBER	TEXT/DESCRIPTION
NULL_ERR	= 0	
GENERAL_ERR	= -1 :	error
INV_OPTION_ERR	= -2 :	option not available
INV_SIGN_ERR	= -3 :	invalid parameter sign
NO_OPEN_ERR	= -4 :	cannot open device
INV_FILE_DATA_ERR	= -5 :	invalid data in file
INV_P_NAME_ERR	= -6 :	invalid parameter name
DUPL_CHAR_ERR	= -7 :	duplicate characters found
DUPL_LUN_ERR	= -8 :	duplicate LUN's assigned
INV_CHAR_ERR	= -9 :	invalid character
DUPL_P_CODE_ERR	= -10 :	duplicate parameter codes
DUPL_P_NAME_ERR	= -11 :	duplicate parameter names
INV_P_CODE_ERR	= -12 :	invalid parameter code
FREE3	= -13 :	no message
INV_DATA_ERR	= -14 :	invalid data
INV_RESPONSE_ERR	= -15 :	invalid response
INV_DEV_ERR	= -16 :	invalid device
FREE4	= -17 :	no message
FREE5	= -18 :	no message
INV_PLOT_POSTN_ERR	= -19 :	invalid plot position
SRC_DEV_ERR	= -20 :	error at source device
DST_DEV_ERR	= -21 :	error at destination device
OUT_RANGE_ERR	= -22 :	value out of range
INV_P_TYPE_ERR	= -23 :	invalid parameter type
NON_UNIQ_DEV_ERR	= -24 :	source same as destination device
INV_DATE_ERR	= -25 :	invalid date
INV_LINE_NBR_ERR	= -26 :	invalid line number
INV_BLK_LEN_ERR	= -27 :	invalid block length
NO_P_CODE_ERR	= -28 :	missing parameter code
SYNTAX_ERR	= -29 :	syntax error
INV_CONST_ERR	= -30 :	invalid constant
TBL_FULL_ERR	= -31 :	table full
NO_ROOM_IN_BFR_ERR	= -32 :	not enough buffer space
TOO_LONG_ERR	= -33 :	text too long for conversion
DIV_ZERO_ERR	= -34 :	divide by zero
LIMIT_ERR	= -35 :	limit error
AMB_SPEC_ERR	= -36 :	ambiguous specification
NO_PARM_ERR	= -37 :	missing parameter(s)
INV_P_LEN_ERR	= -38 :	invalid parameter length



APPENDIX D

FIELD CHECKING SYSTEM HARDWARE CONFIGURATION

D.0	SCOPE	page 1
D.1	MEMORY-MAPPED ADDRESSES	2
D.2	MEMORY PARTIONS	2
D.3	INTERRUPT ASSIGNMENTS	2
D.4	DMA-CHANNEL ASSIGNMENTS	2



D.0 SCOPE

This Appendix describes the hardware configurations of the IBM-AT as they apply to the Field Checking System. Any configurations not specified in this appendix remain as they are defined in the Technical Reference Personal Computer AT Manual, IBM document #1502243.

Section D.1 lists the memory-mapped addresses used by the Field Checking System. Section D.2 outlines the memory portions of the System. Section D.3 lists any interrupts used and the devices to which they are attached. Section D.4 lists the DMA channels used by the System.



D.1 MEMORY-MAPPED ADDRESSES

ADDRESS	DESCRIPTION
>0340	PC-2 Iomega Disk Drive Controller
>03C0	TC50M Tape Controller I/O

D.2 MEMORY PARTIONS

RANGE	DEDICATION
512K-640K	TC50M Tape Controller (hard addressed) (optional with controller card -- option not purchased)

D.3 INTERRUPT ASSIGNMENTS

INTERRUPT LEVEL	DEVICE
4	TC-50M Tape Controller
5	PC-2 Iomega Disk Drive Controller
7	Versatec Printer/Plotter

D.4 DMA-CHANNEL ASSIGNMENTS

CHANNEL	ASSIGNMENT
1	TC-50M Tape Controller
2	PC-2 Iomega Disk Drive Controller

