LA-11327-MS

UC-66b
Issued: June 1988

LA--11327-MS

DE88 012950

Initial Borehole Acoustic Televiewer Data Processing Algorithms

Troy K. Moore

MASTER

DISTRIBUTION OF THIS DOCUMENT IS UNLIMITED

Los Alamos Los Alamos National Laboratory
Los Alamos, New Mexico 87545

DISCLAIMER

This report was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government nor any agency Thereof, nor any of their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or any agency thereof. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof.

DISCLAIMER

Portions of this document may be illegible in electronic image products. Images are produced from the best available original document.

Table of Contents

FIGURES and TABLE vii

ABSTRACT 1

I. Introduction 1
- A. Motivation 1
- B. Basic Assumptions 1

II.BDCOMP (BAT Data Complement and Fix) 3
- A. Functional Description 3
- B. Implementation 3
- C. BDCOMP User's Guide 5

III. DATASEP (BAT Data Separation) 5
- A. Functional Description 5
- B. Implementation 5
- C. DATASEP User's Guide 16

IV. DPDUMP (DP File Dump) 16
- A. Functional Description 16
- B. Implementation 16
- C. DPDUMP User's Guide 16

V. BDINVRT (BAT Data Invert Utility) 18
- A. Functional Description 18
- B. Implementation 18
- C. BDINVRT User's Guide 18

VI. BDEXTRCT (BAT Data Extraction Utility) 18
- A. Functional Description 18
- B. Implementation 18
- C. BDEXTRCT User's Guide 21

VII. BATD (BAT Display Utility) 21
- A. Functional Description 21
- B. Implementation 21
- C. BATD User's Guide 26
  - 1. Menu Description 26
  - 2. Selecting Records 31
  - 3. Logical Display 31

VIII. BATT (BAT Trace Generation Utility) 34
- A. Functional Description 34
- B. Implementation 34
- C. BATT User's Guide 38
  - 1. Menu Description 38
  - 2. Display Definition 42
  - 3. Usage Notes 47

REFERENCES 47

Table of Contents

APPENDICES

A. Workstation Development Environment 49
B. Disk Files Used to Generate Algorithms 50
C. Field Tape Format 51
D. Typical Terminal Session Output 53
E. BDCOMP Source Code 55
F. DATASEP Source Code 59
G. DPDUMP Source Code 77
H. BDINVRT Source Code 81
I. BDEXTRCT Source Code 85
J. BATD Source Code 90
K. BATT Source Code 138

Figures

Figure	1.	Typical Processing Path	2
Figure	2.	BDCOMP Flowchart	4
Figure	3.	Files Produced by DATASEP	6
Figure	4.	DATASEP Flowchart	7
Figure	5.	file_prep Flowchart	9
Figure	6.	name_gen Flowchart	10
Figure	7.	find_first_rec Flowchart	10
Figure	8.	find_next_rec Flowchart	11
Figure	9.	get_depth Flowchart	12
Figure	10.	save_log_heads Flowchart	12
Figure	11.	Log Header File Format	13
Figure	12.	Data File Format	14
Figure	13.	DP File Format	15
Figure	14.	DPDUMP Flowchart	17
Figure	15.	BDINVRT Flowchart	19
Figure	16.	BDEXTRCT Flowchart	20
Figure	17.	BATD Flowchart	22
Figure	18.	rec_sel_rec1 Flowchart	24
Figure	19.	rec_sel_rec2 Flowchart	25
Figure	20.	depth_sel_rec Flowchart	27
Figure	21.	disp_recs Flowchart	28
Figure	22.	BATD Menu Structure	29
Figure	23.	Example of Display Generated by BATD	32
Figure	24.	Parameters Used to Generate Figure 23	33
Figure	25.	BATT Flowchart	35
Figure	26.	select_recs Flowchart	37
Figure	27.	intrvl_calc Flowchart	39
Figure	28.	calc_tick_lines Flowchart	40
Figure	29.	gen_disp Flowchart	41
Figure	30.	BATT Menu Structure	43
Figure	31.	Typical Traces Generated by BATT	43
Figure	32.	Parameters Used to Generate Figure 31	44
Figure	33.	Example of Two Traces with Associated Tick Lines	45

Table

Table	I.	Tick Line Representations	46

Initial Borehole Acoustic Televiewer Data Processing Algorithms

by

Troy K. Moore

ABSTRACT

With the development of a new digital televiewer, several algorithms have been developed in support of off-line data processing. This report describes the initial set of utilities developed to support data handling as well as data display. Functional descriptions, implementation details, and instructions for use of the seven algorithms are provided.

I. Introduction

A. Motivation

Recently, a new borehole acoustic televiewer has been developed jointly by Westfalische Berggewerkschaftskasse (WBK) of Bochum, FRG, and Los Alamos National Laboratory.[1,2] This tool records digital data collected in the field on 1/4-in. data cartridges. Once the field data have been read into a processing workstation, various mission-specific processing algorithms may be applied.

This report documents the algorithms developed for initial off-line processing of Borehole Acoustic Televiewer (BAT) data. The function and implementation of each algorithm are described. A user's guide for each is also included. Figure 1 illustrates the processing paths provided by the utilities described herein. Algorithms are discussed in the order they could be encountered in a processing scenario.

B. Basic Assumptions

Several basic assumptions and preconditions should be defined before examining each utility. These include the following:

1. This document is intended to be an overview of the currently existing utilities. Development environment details are contained in Appendices A - F. For exact details such as passed parameters, variable names, etc., the user is directed to the in-code documentation. See Appendices E - K.

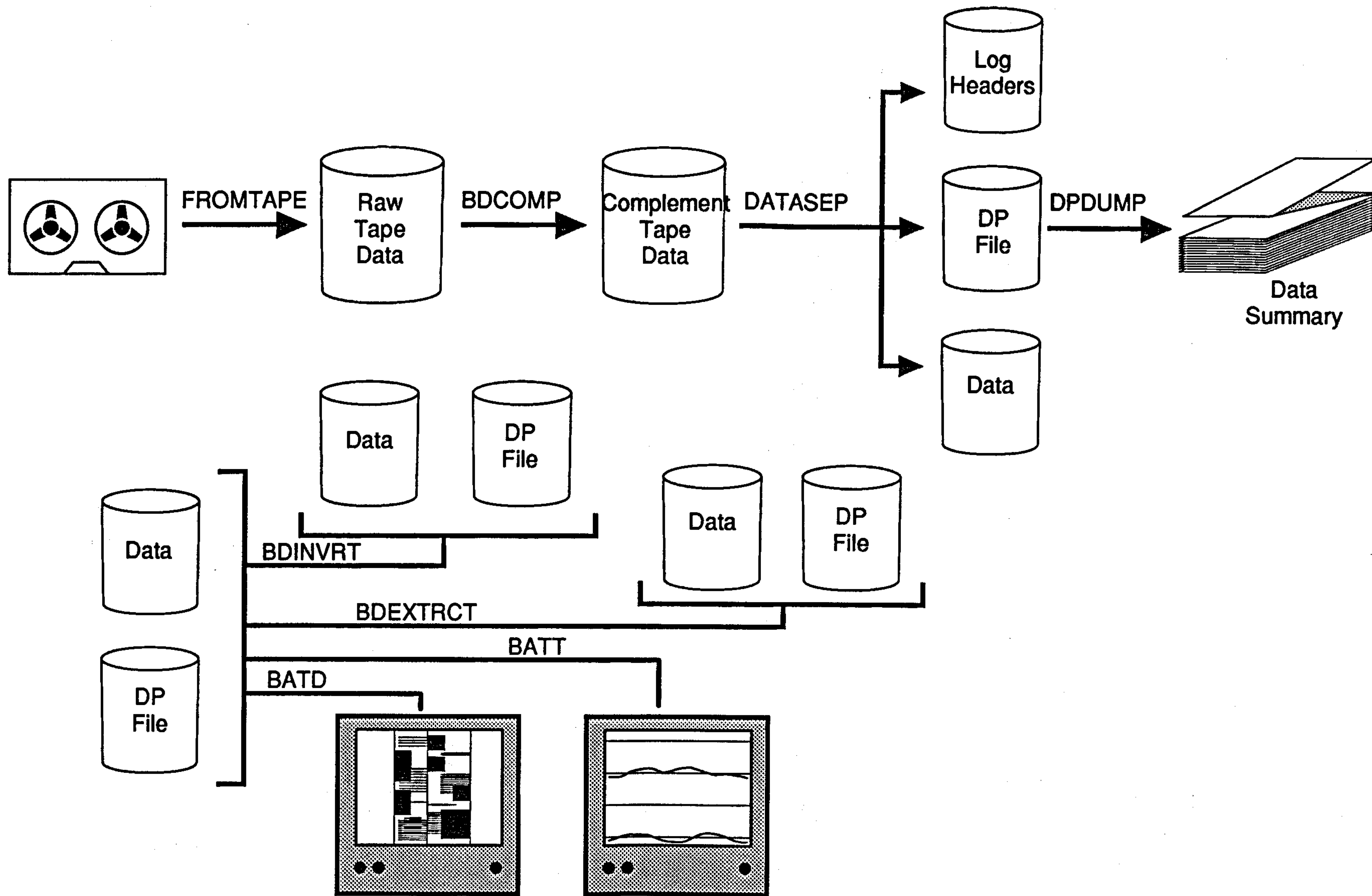

Figure 1. Typical processing path.

2. Console refers to the CRT used as the primary output device connected to the CPU, whereas display is the monitor attached to the imaging board set.
3. Unless overridden, data from shallower portions of the area of interest will be displayed at the top of the display. As the eye travels down the display, data from deeper in the wellbore will be encountered.
4. When using BATD and BATT, all data are assumed to have been collected as the tool is extracted from the wellbore. This will be considered the standard data format.
5. In all descriptions except that for BATT, the terms travel time and caliper can be used interchangeably.
6. Offsets begin with 0; record numbers start with 1.
7. The tape transfer utility FROMTAPE will not be discussed in this report. This utility was purchased and has its own documentation.
8. Utilities described in the report are still being developed; they may change--and contain errors.

II. BDCOMP (BAT Data Complement and Fix)

A. Functional Description

The initial data collection of April 13, 1987, used downhole and uphole software that contained two unique features:

- the *S* in the log header keystring *Sabis Field Tape* was replaced with 00h, and
- all data were complemented before being written to tape.

BDCOMP was developed to complement the data read from the field tapes and replace the missing *S*. Actual field tape format is shown in Appendix C.

B. Implementation

BDCOMP was written as a single function designed to process one buffer of data at a time. See Figure 2 for a detailed flowchart. Each byte not equal to 0FFh (00h complemented) is automatically complemented. Any 0FFh bytes are treated as though they may represent a missing *S*. To determine where the *S* should be inserted, a four-character subkeystring following the current buffer position is examined. If this subkeystring contains the complement of *abis*, then an *S* is inserted in the current position. Even though this is not an absolute test (an absolute test would require matching the entire subkeystring *abis Field Tape*), the existence of 0FFh followed by the complement of *abis* is sufficient evidence to insert the missing character.

Note that future versions of the uphole software will most likely have the *S* removal bug corrected. BDCOMP will work correctly in either case as long as the data still need to be complemented.

Implementation is straightforward except for identifying a potential substring that is not entirely contained within the buffer. In this case, the file pointer must be manipulated in preparation for the next read operation. Statistical information provided at the termination of BDCOMP indicates the number of file repositions required as well as the actual number of *S*'s inserted.

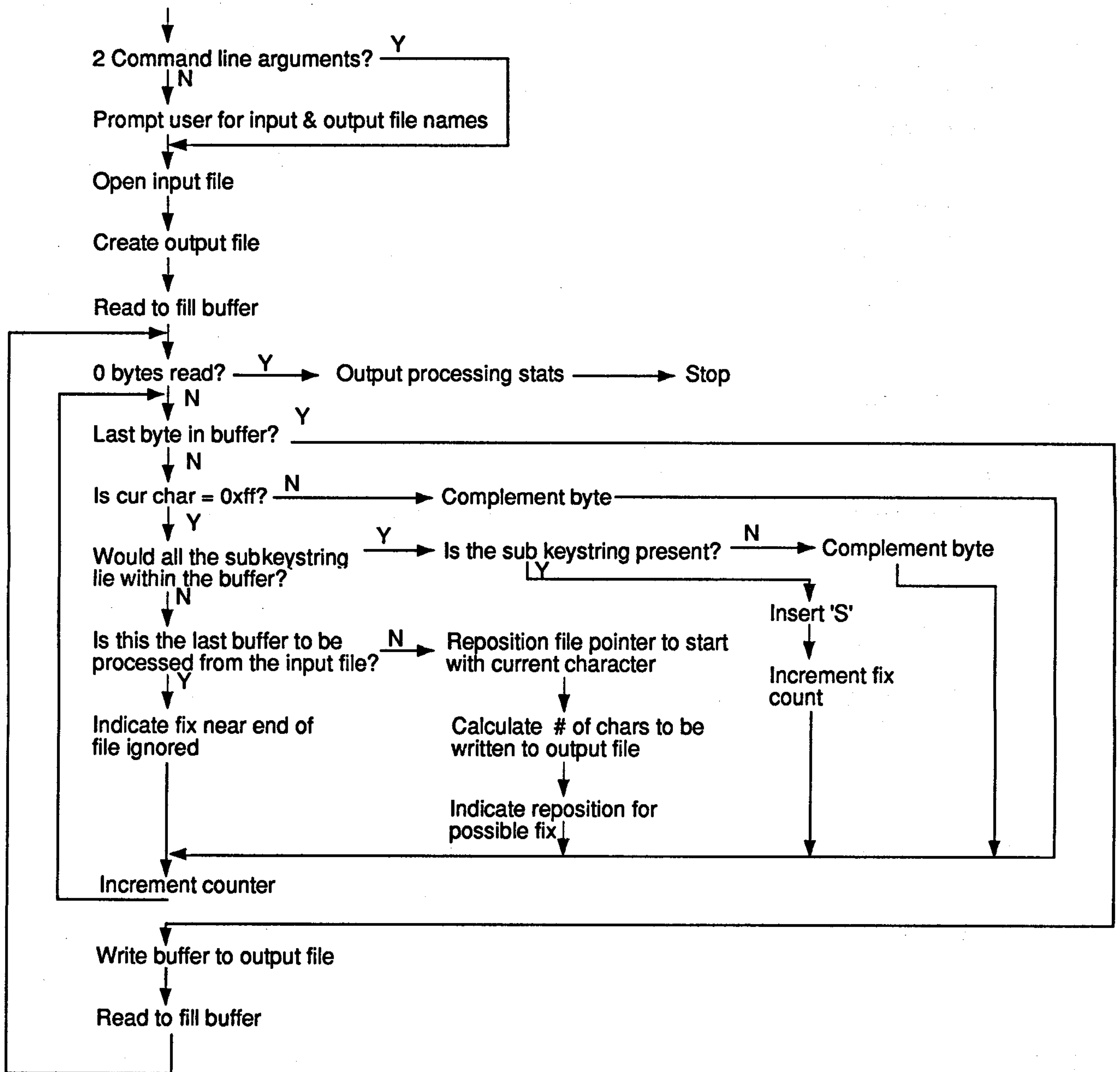

Figure 2. BDCOMP flowchart.

C. BDCOMP User's Guide

Input file and output file specifications are the only parameters required for operation. These parameters may be entered by the command line or in response to run-time prompts. Drive and path specifications are supported.

The command line syntax is as follows:

BDCOMP input_file output_file

III. DATASEP (BAT Data Separation)

A. Functional Description

DATASEP is designed to separate a corrected raw tape data file (post-BDCOMP processing) into three separate files. This operation is included to ease data handling in subsequent processing. The first file produced represents all log header blocks located within the tape file. A second file contains only revolution data (revolution header, amplitude, and travel time data). The third file, referred to as the data pointer (DP) file, provides random access to data contained within the second file. See Figure 3.

B. Implementation

DATASEP classifies a sequence of bytes from the input file as either a log header record or a revolution data record based on keystrings. These keystrings are *Sabis Log Header* for log header records or *Sabislog* for revolution data. Once a keystring has been located, the next keystring is found to determine record length. A high-level flowchart is shown in Figure 4.

DATASEP is implemented as a moderately complex main routine calling six functions. All definitions and functions are contained within the file DATASEP.C. Each function is described as follows:

1. file_prep

   Obtain the input file name from the user. A set of candidate output file names is generated. If unacceptable, the user can enter new output file names. All files are then opened or created.

2. name_gen

   Called by file_prep to generate default output file names.

3. find_first_rec

   Used at startup to find the first record in the input file. This function calls find_next_rec.

4. find_next_rec

   Find and classify the next keystring in the buffer. When this information is used, the end of the current record, start of the next record, and type of next record can be determined.

5. get_depth

   Generates a floating point depth value from the ASCII depth string in the revolution data header.

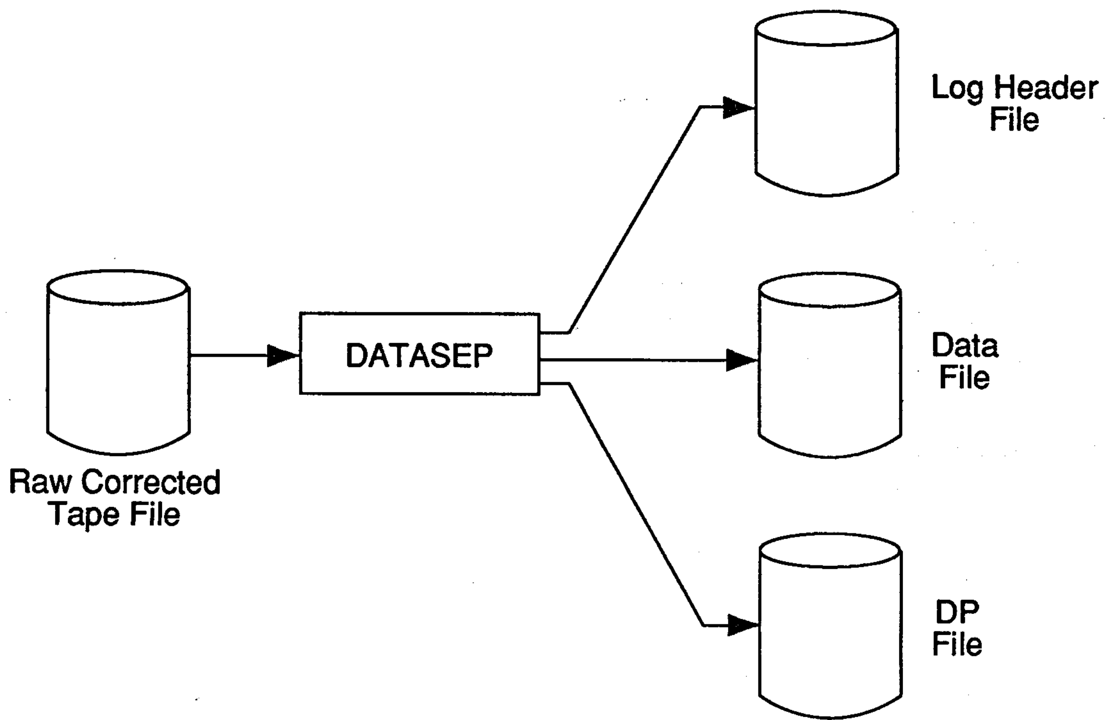

Figure 3. Files produced by DATASEP.

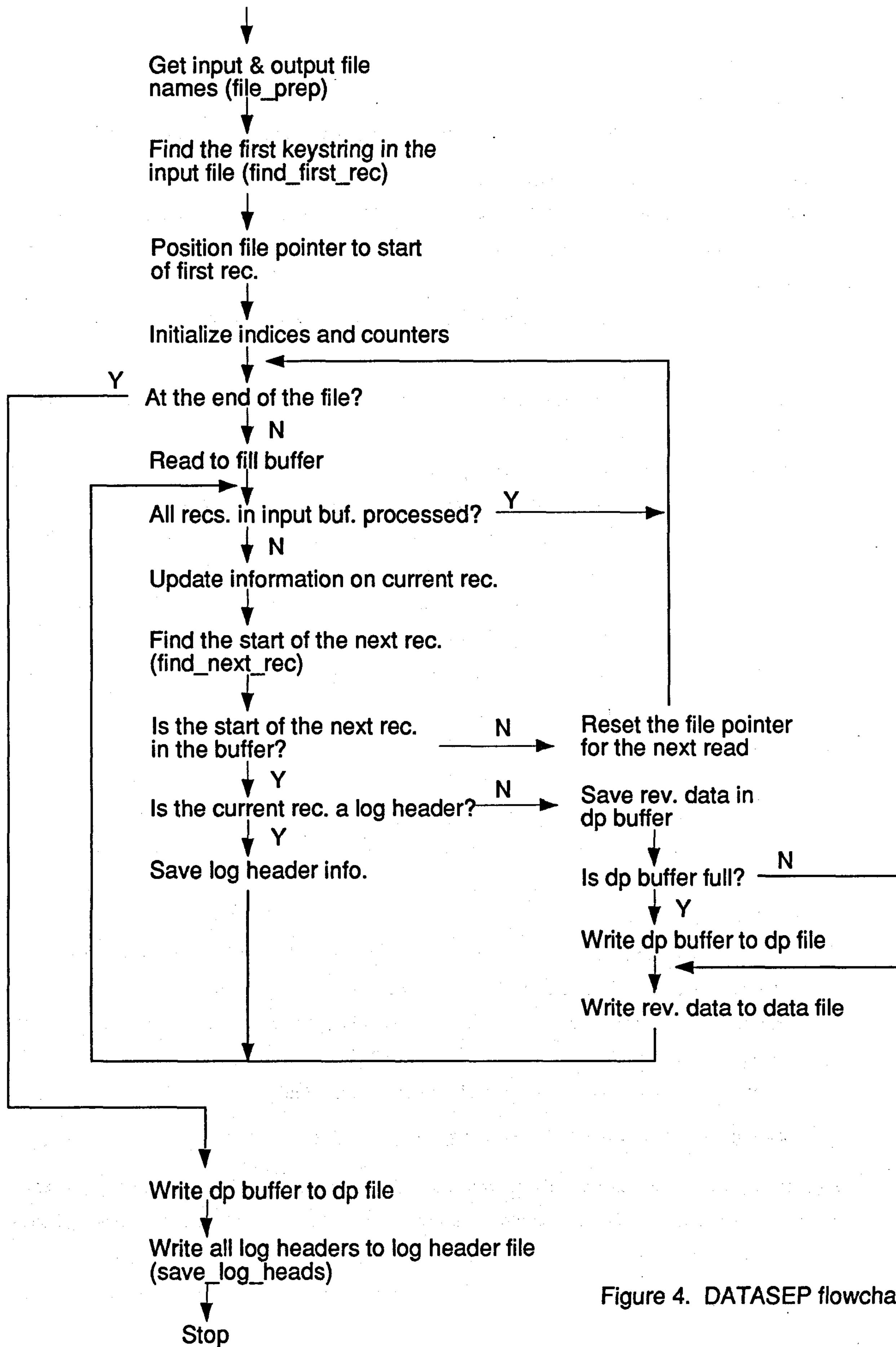

Figure 4. DATASEP flowchart.

6. save_log_heads

Writes log header records and associated location and length information to the log header file.

Figures 5 - 10 contain flowcharts for each function.

Several pointers and indices are used throughout DATASEP to keep track of the current location in the input buffer as well as several data structures. The names and uses of these variables are documented within the code. See Appendix F.

DATASEP generates three output files, each with a unique format. These files are discussed below.

1. Log header file

This file will contain all the log header records found while processing the input file. It is possible to have several different log headers in a single tape file.

To determine where in the input file the log header record was located and to know how long the log header record is, six bytes of information are appended to the start of each header record. An offset (long int) relative to the start of the input file is contained within the first four bytes of each log header record. The remaining two bytes (int) represent the actual length of the log header record in the input file. When existing WBK software is used, this length is 1024 bytes. Therefore, the length of each record in the log header file should be

log header length + appended info = 1024 + 6 = 1030 bytes.

The default file extension for the log header file is LHF. See Figure 11.

2. Data file

This file is similar to the input file format except that all the log header information has been removed. Any gaps produced by removing log headers are filled by sliding the revolution data forward. See Figure 12. No additional information is appended.

The default file extension for this file is RAW.

3. Data pointer file

The DP file provides random access to any revolution data record found in the data file. Each record in this file contains three entries. For the nth entry in the DP file:

record offset	- byte offset from the start of the data file to the start of the nth revolution (long int)
depth	- depth at which revolution n was collected (float)
length	- number of bytes in revolution data record n (int)

DPF is the default file extension for the DP file. See Figure 13.

Log header and revolution data records are written to their respective output files in vastly different ways. Each revolution data record is written to the data file as it is encountered. If there are no more than a few log header records per input tape file, the log header records are queued until all main control loop processing is complete. At that time, all log header records are appended auxiliary information and are written to the output file. The

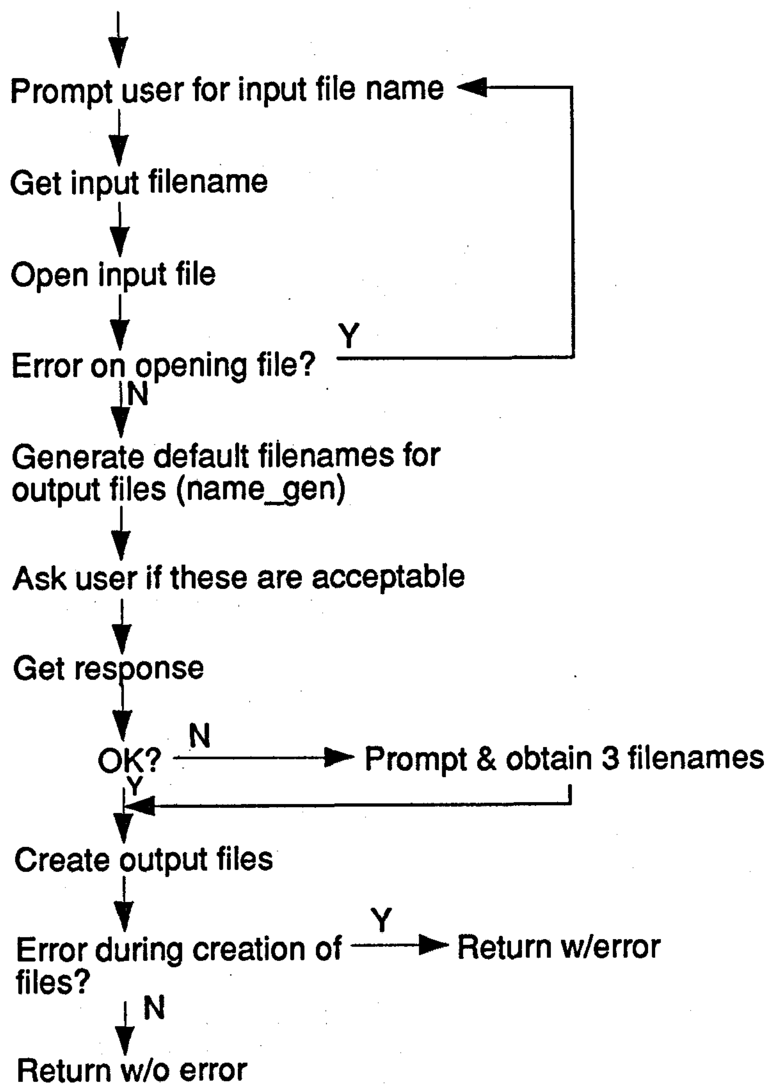

Figure 5. file_prep flowchart.

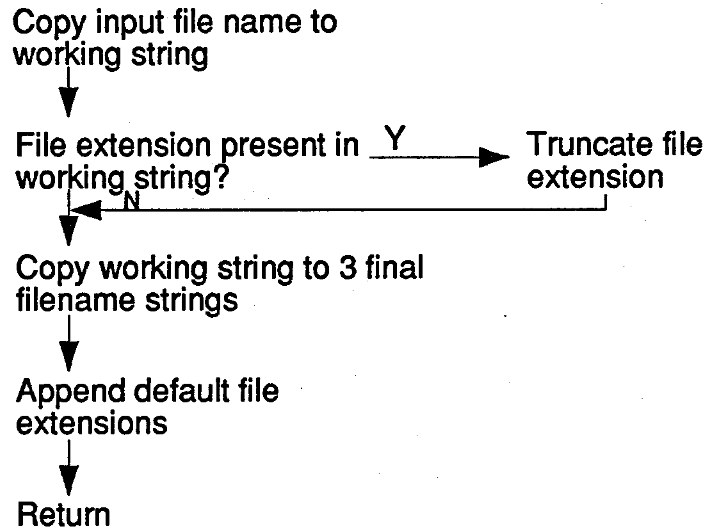

Figure 6. name_gen flowchart.

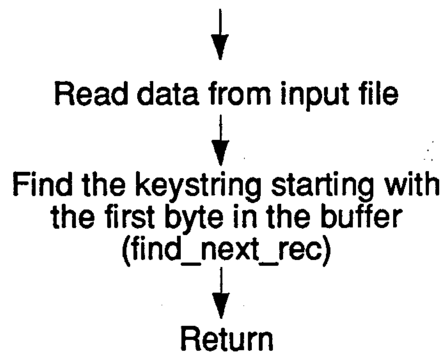

Figure 7. find_first_rec flowchart.

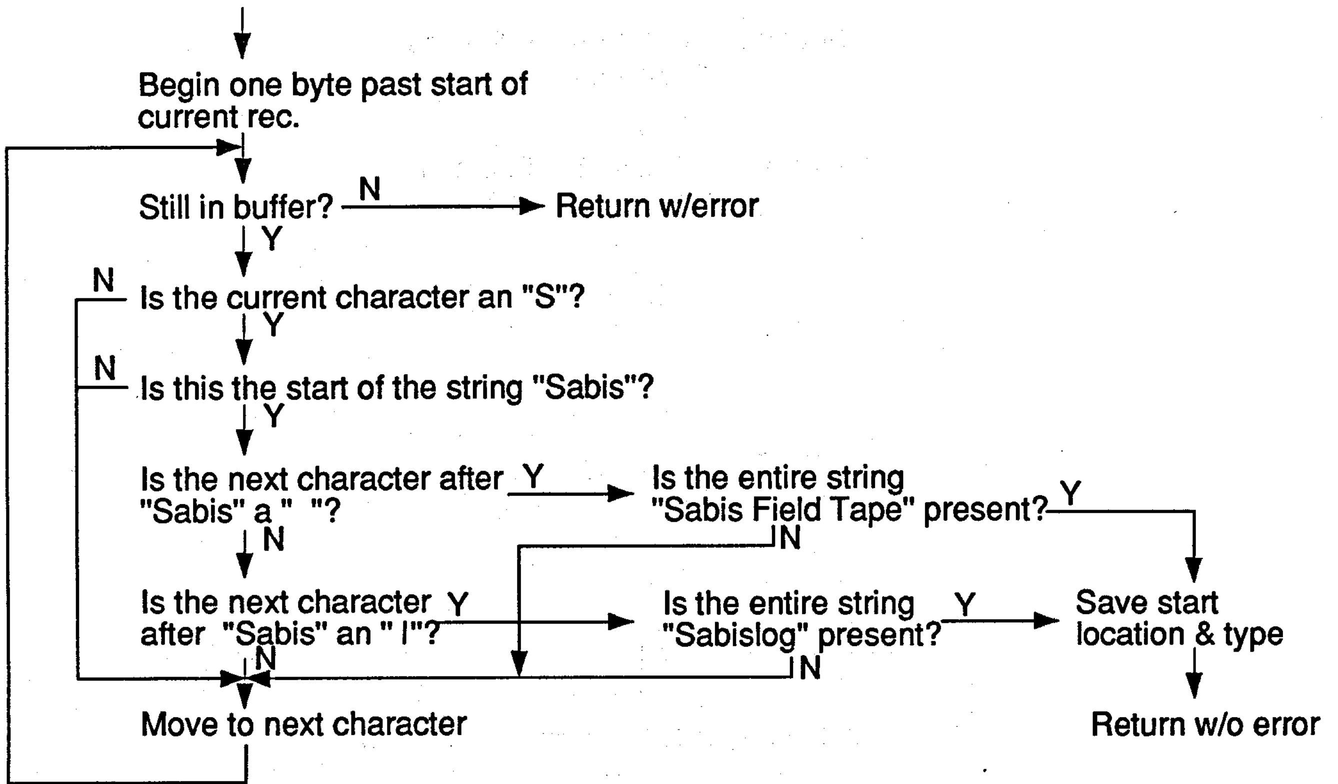

Figure 8. find_next_rec flowchart.

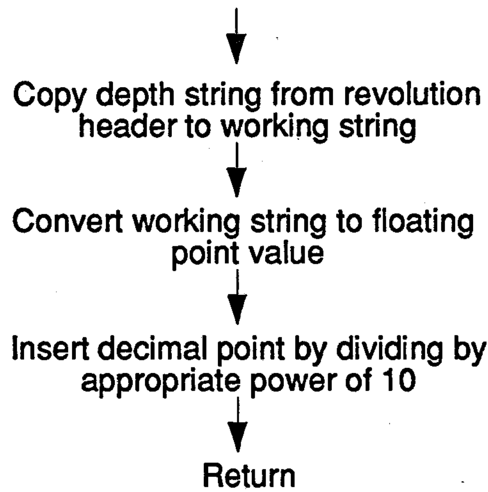

Figure 9. get_depth flowchart.

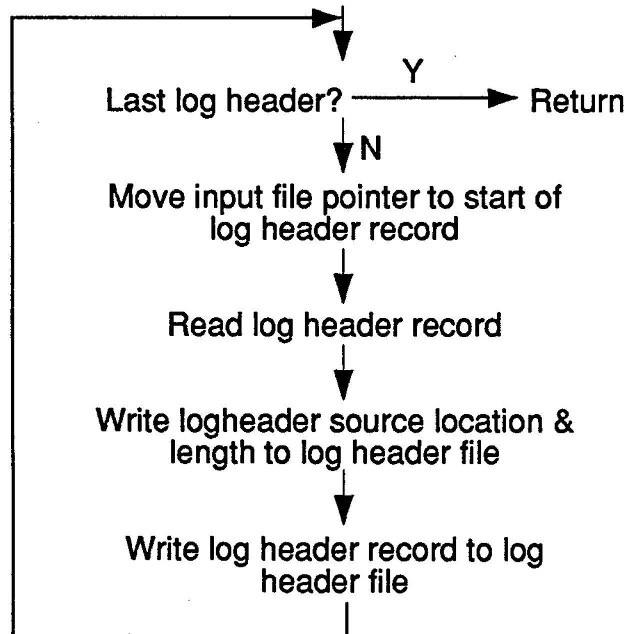

Figure 10. save_log_heads flowchart.

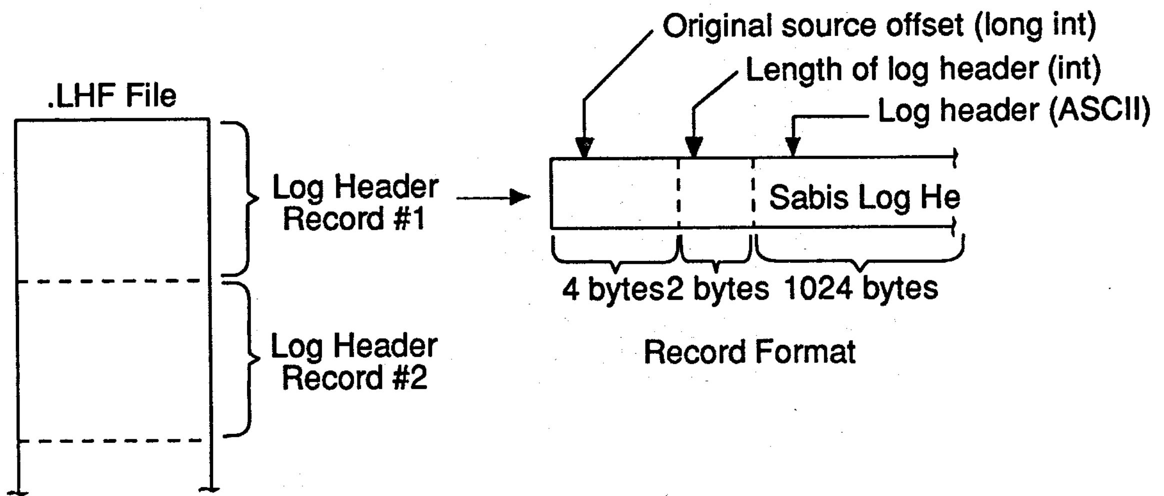

Figure 11. Log header file format.

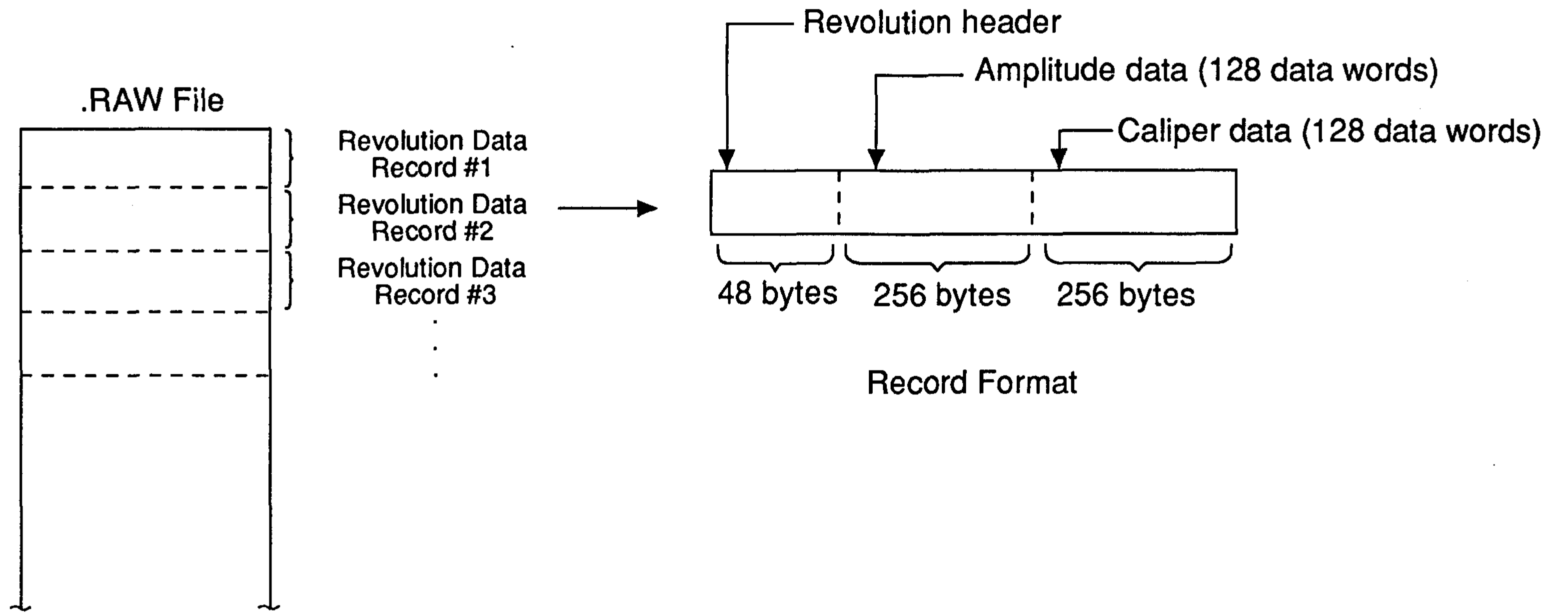

Figure 12. Data file format.

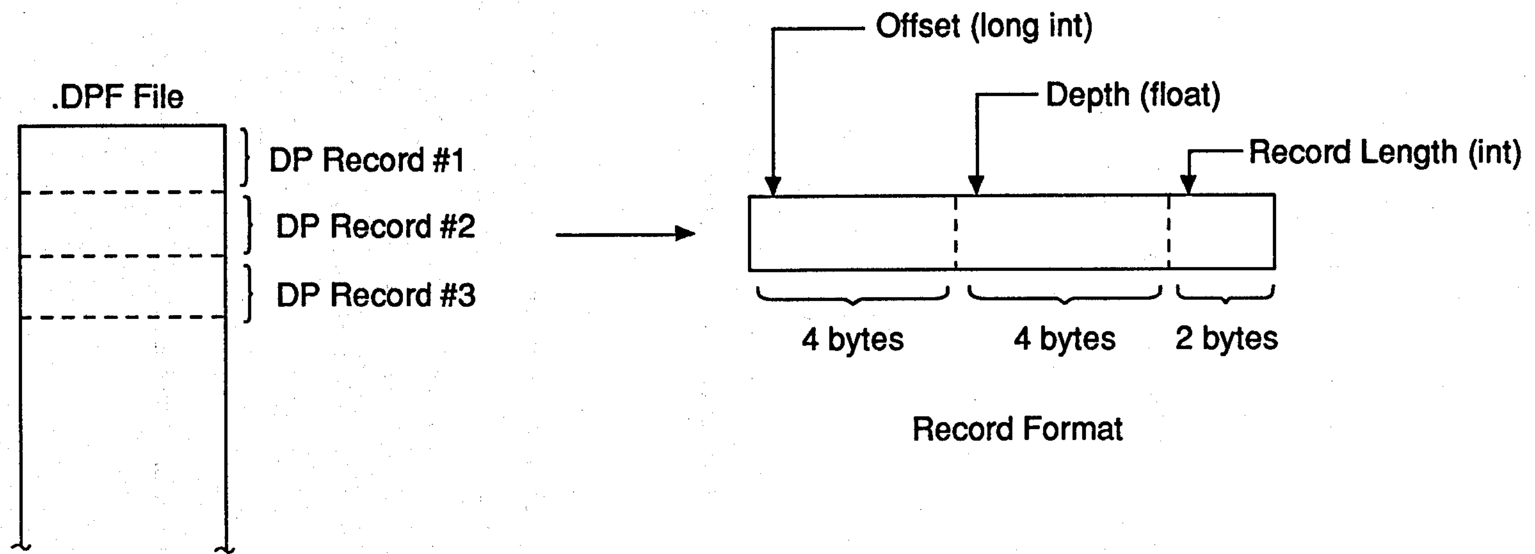

Figure 13. DP file format.

possibility that the number of log headers may exceed the queue size does exist, but it is extremely unlikely. DATASEP (Version 1.11) allows 64 queued log headers.

As DATASEP is terminating execution, a processing summary is output to the console. See Appendix D. This summary includes the number of log headers and revolution data records identified. Note that the number of records may be one less than expected because the final record in the file does not have a keystring following it for determining the end of the record.

C. DATASEP User's Guide

If the user accepts the algorithm's default output files, the only user input is the input file name. Overriding the default output names is accomplished by striking any key other than Enter, which allows entry of three different user-specified filenames with any file extension.

IV. DPDUMP (DP File Dump)

A. Functional Description

Once the data and DP files have been generated, it is convenient to summarize data file contents. The utility DPDUMP has been developed to meet this need. This utility prints the offset, depth, and length of each data file record as found in the DP file. For reference, a record number is generated and output with each DP file entry. In addition, records that are too long (not within a user-specified range) are flagged as possibly invalid. Figure 14 contains a high-level flowchart.

B. Implementation

The implementation of this utility is straightforward and simple enough that everything is handled within the main function. Output is produced in a format suitable for the printer even though the data stream is sent to *stdout*. Records that are too long (not within the specified range) are appended an * when output.

Output is generated on a page-by-page basis. Because there are two columns of information per page, handling the final page is the only concern. If possible, the first column will be filled completely before starting on the second.

C. DPDUMP User's Guide

DPDUMP is a command line arguments-only interface requiring three parameters. The syntax is as follows:

DPDUMP input_file base deviation

The base and deviation are used in identifying records whose length does not meet the following condition:

base - deviation <= record length <= base + deviation

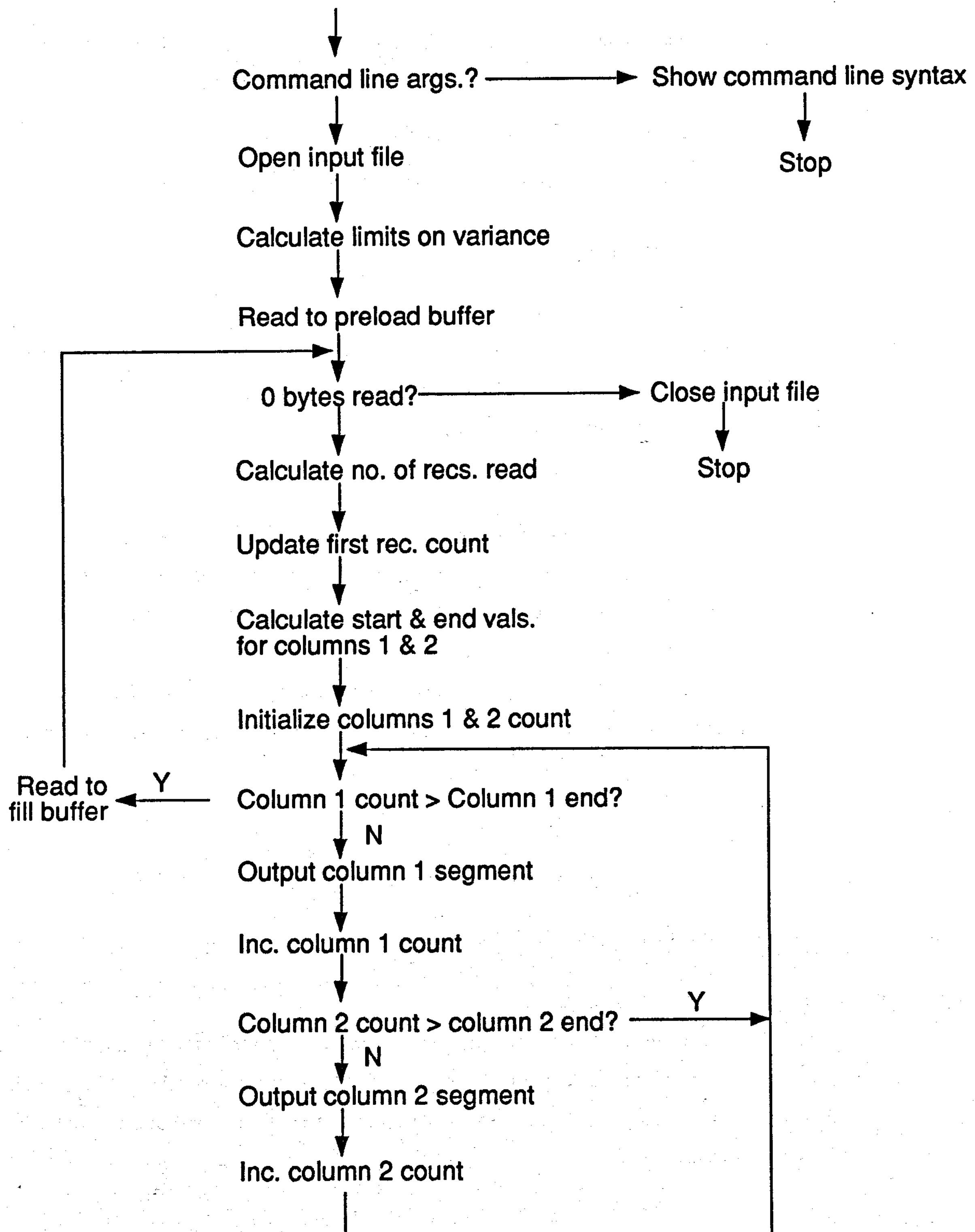

Figure 14. DPDUMP flowchart.

As previously indicated, the output produced by DPDUMP is suitable for output to the printer even though it, by default, is sent to the console. To redirect the output to the printer, use the following syntax:

DPDUMP input_file base deviation > PRN

Typical output produced by DPDUMP can be seen in Appendix D.

V. BDINVRT (BAT Data Invert Utility)

A. Functional Description

As indicated in Section I.B., it is assumed that data are always collected as the tool is extracted from the wellbore. This assumption may not always be true. BDINVRT was developed to invert data collected as the tool is lowered into the borehole into the standard data format.

B. Implementation

BDINVRT is an interactive algorithm implemented as a single function. A set of DP and raw data files is used as input. New DP and data files are produced that contain the original data but in an inverted order. The first revolution record is now the last. Data are processed on a revolution-by-revolution basis starting with the last revolution in the input file set. See the Figure 15 flowchart.

The record offsets in the new DP file are referenced to the offset from the start of the first byte in revolution records found in the new raw data file. The accelerometer data found in the revolution header are not modified during execution.

C. BDINVRT User's Guide

BDINVRT prompts the user for input. Only the input and output DP and raw data file names are required.

VI. BDEXTRCT (BAT Data Extraction Utility)

A. Functional Description

During data collection, there are times when the tool is stationary or the data may be invalid. Data acquired during such periods do not contain any additional information. Because a smaller data set results in increased processing speed, data of the greatest interest should be extracted from the original data set. BDEXTRCT has been developed to meet this need.

B. Implementation

BDEXTRCT is an interactive algorithm implemented as a single function. An existing DP and raw data file set is used as input. A new set of DP and data files is produced that is a subset of the input. The user is prompted for input files, output files, and first and last records specification. Upon positioning the DP file pointer at the first requested record entry, data are processed on a revolution-by-revolution basis until the final requested record is reached. See Figure 16.

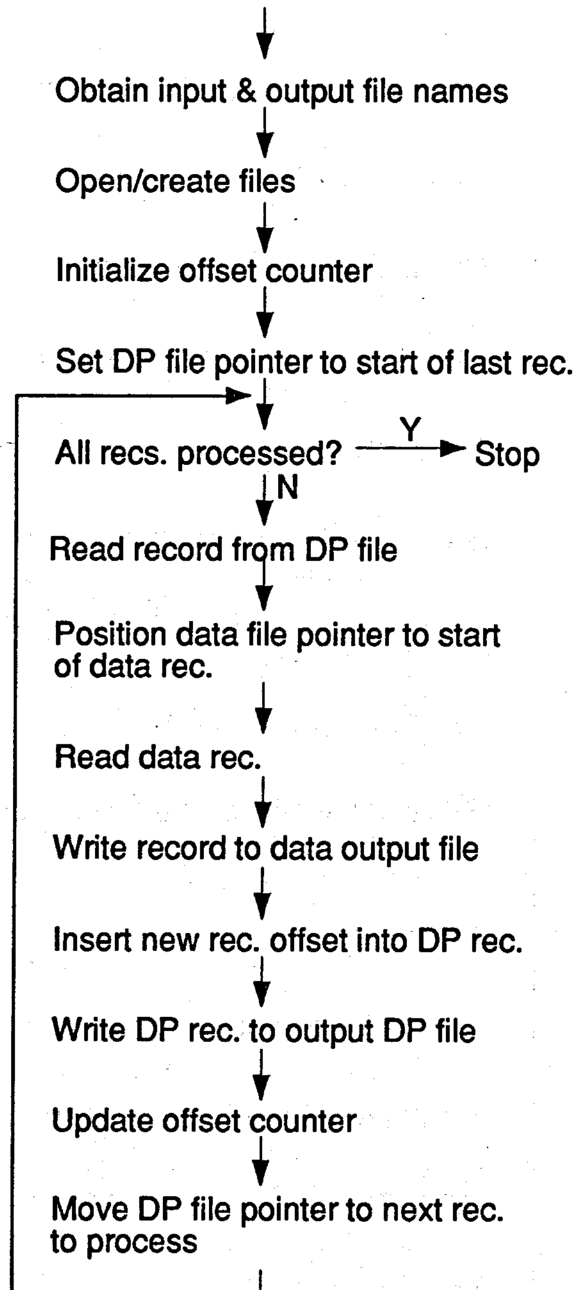

Figure 15. BDINVRT flowchart.

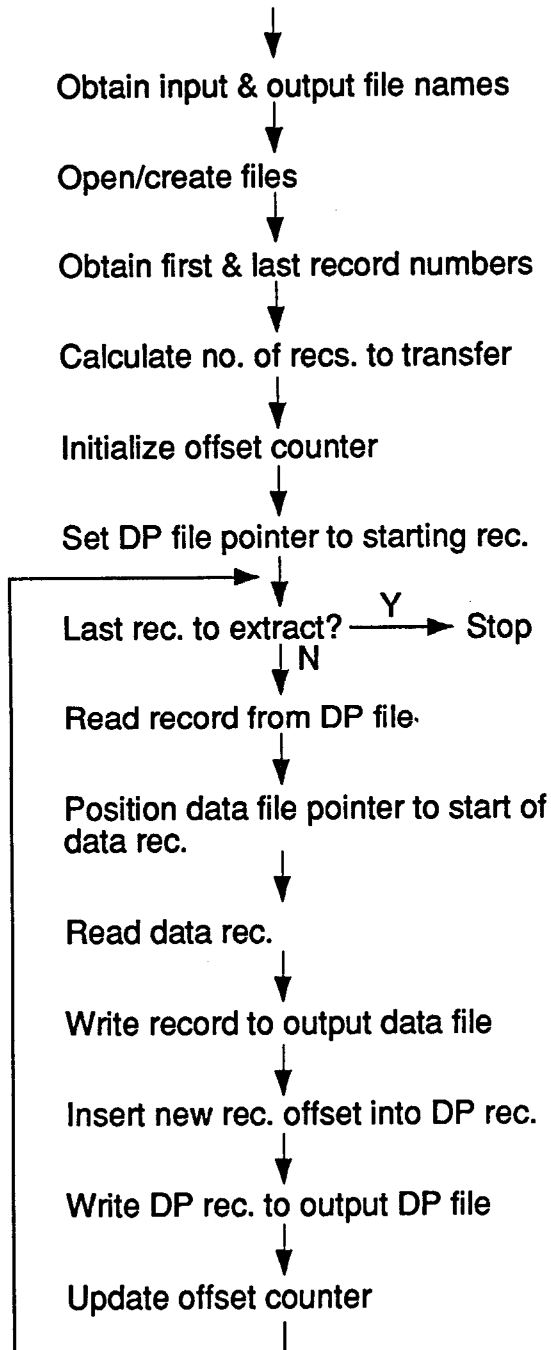

Figure 16. BDEXTRCT flowchart.

The record offsets in the new DP file are referenced to the first byte of revolution records found in the new raw data file.

C. BDEXTRCT User's Guide

Three sets of information must be specified by the user. The initial input required is the DP and raw data file from which the extraction will occur. The second input specifies the file names of the files to be created. Initial and final records to be extracted represent the final data required. Records are referenced beginning with 1.

VII. BATD (BAT Display Utility)

A. Functional Description

BATD is a utility for displaying BAT tool data. Either amplitude, or travel time, or both may be displayed. Scale factors for each type of data are specified by the user. The output may be displayed either in Channel 0 or in Channel 1 of the image memory. The user can specify where output is to be positioned on the display. Records to be displayed may be selected on a record or depth basis. Two methods are available for specifying the range of records to be selected. Extensive parameter checking helps the user to enter only valid parameters.

This utility is menu driven with several levels of pull-down submenus to provide the user with a friendly interface. Data forms[3] are used to obtain parameters from the user. A status window is constantly updated to provide the user with current parameters. Any additional information or error messages are relayed to the user through windows.[4]

B. Implementation

BATD is a complex algorithm simplified by the extensive use of functions. The main routine is predominantly a collection of function calls, which break the complex algorithm up into more understandable sub-tasks. Figure 17 provides an overview of the algorithm.

Two special libraries of functions were used to develop this code. The first, *Windows for Data*, Vermont Creative Software, was used to design/implement the menu structure. This library was also used to provide informational windows as well as data forms for entering data. Access to the image processing boards was provided by the DT-IRIS library from Data Translation.[5]

To avoid passing numerous parameters between functions, global variables are used extensively. Information pertaining to these variables as well as to all the *#defines* is contained in a set of include files. A directory of all files used is included in Appendix B.

To provide insight into the concepts involved in the implementation of this algorithm, each of the functions developed for BATD will be discussed in the following pages. Flowcharts are provided for the more involved functions.

1. init_parms

Initializes many of the global parameters to default values.

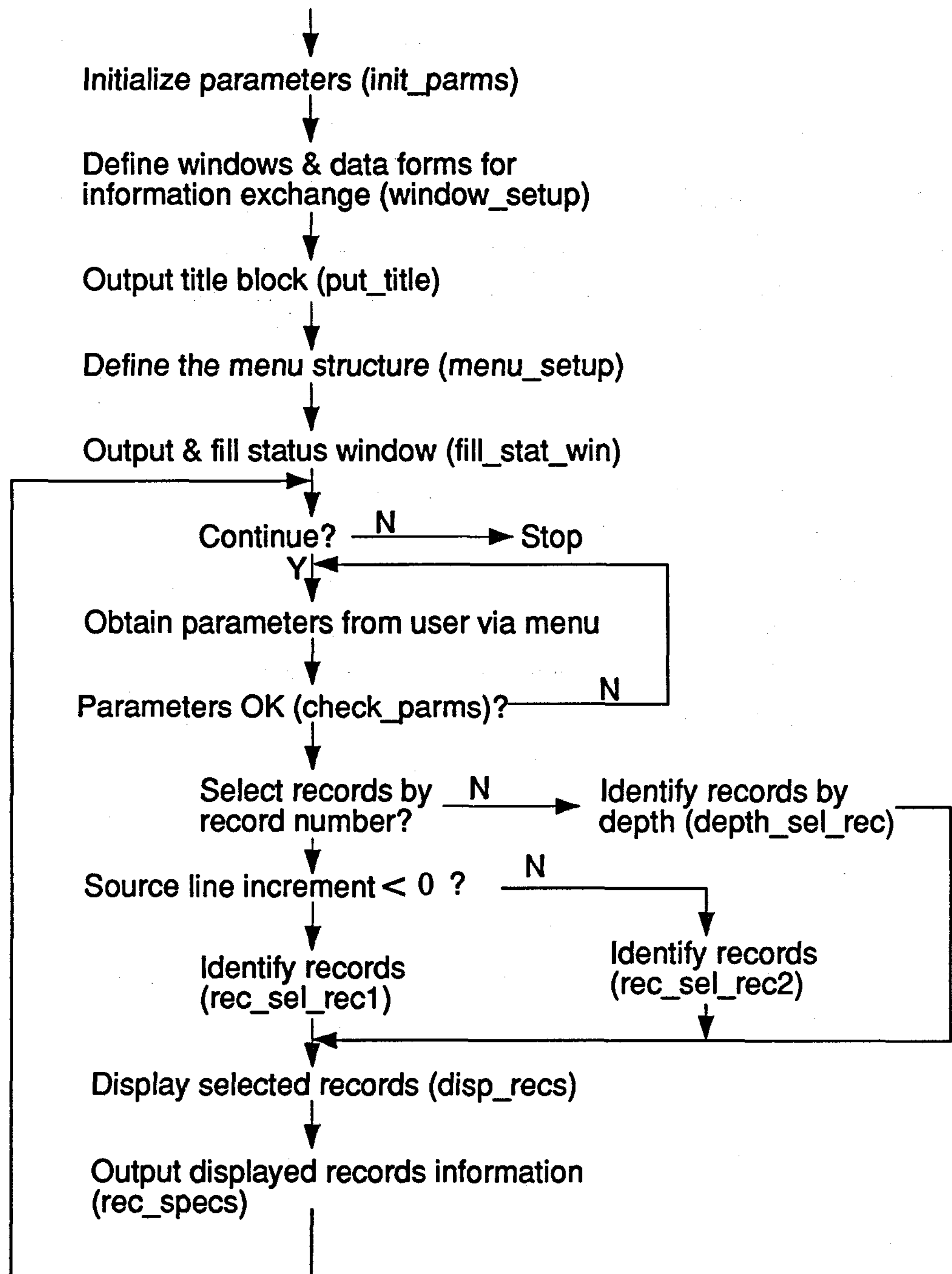

Figure 17. BATD flowchart.

2. window_setup

   Defines the windows and data forms that provide information to and obtain parameters from the user.

3. put_title

   Places the title window on the console at startup.

4. menu_setup

   Defines the menu structure including those action functions called as the result of a menu item being selected.

5. fill_stat_wn

   At startup, fills the status window with parameter values by repeated calls to the function stat_wind_wrt. stat_wind_wrt updates a specific line in the status window and is called any time a parameter changes.

6. Action functions

   When a bottom-level menu option is selected, 1 of 14 action functions will be called. These functions will result in obtaining parameter(s) from the user, selecting an option, exiting the menu structure to display data, or terminating the algorithm.

7. check_parms

   Checks all user-entered parameters before the algorithm is permitted to continue. If any invalid parameters are located, the user is notified and returned to the menu structure.

8. error_handler

   Provides a single place for all error messages to be processed. All error messages are output through a pop-up window.

9. rec_sel_rec1

   This function is used to extract records from the data set when the starting record number is greater than the final record. See the Figure 18 flowchart. A data structure containing the offset and record length of the selected records is returned to the calling procedure. Only the DP file is required to determine which records to use.

   With the standard data orientation, this function begins with the final specified record in order to move forward through the buffer.

10. rec_sel_rec2

    This function is similar to rec_sel_rec1 except for two differences: rec_sel_rec2 is used only when the starting record number is less than the final record, and it begins determining selected records with the start record. See Figure 19.

    rec_sel_rec1 and rec_sel_rec2 were implemented as two separate functions to simplify the logic.

11. depth_sel_rec

    This function is considerably different from the two rec_sel_rec? functions described above, even though its task is also to select records for display. It is used when the beginning and ending depths

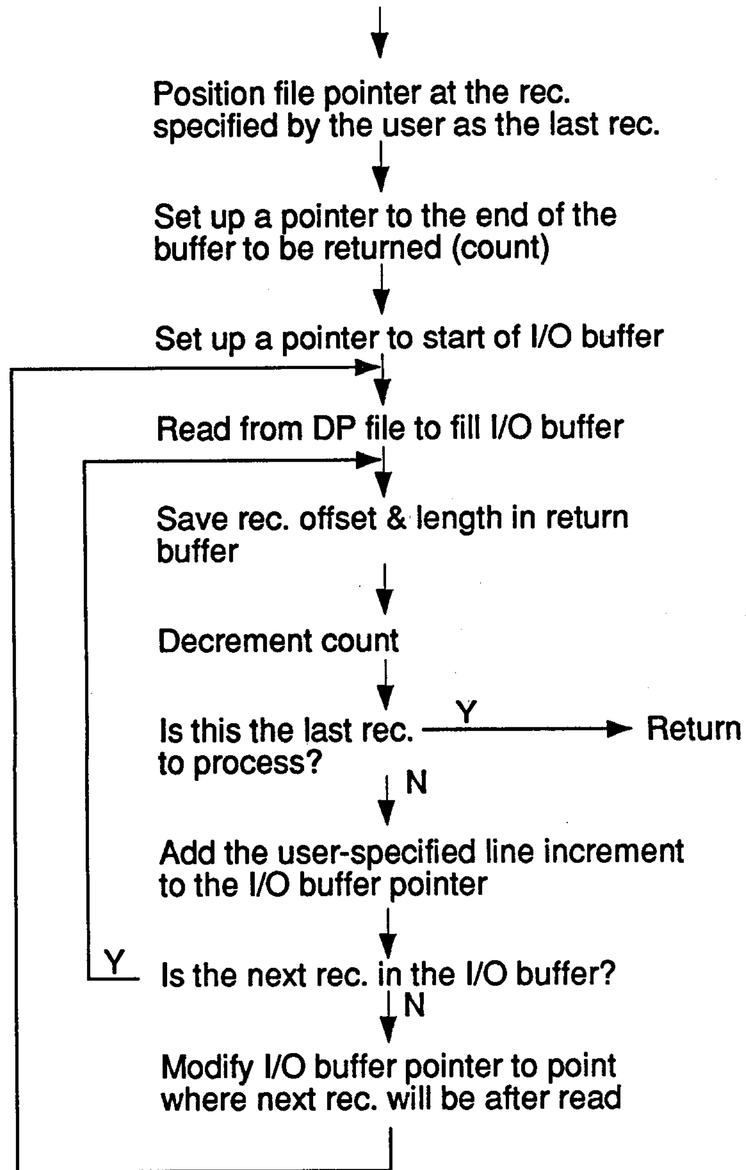

Figure 18. rec_sel_rec1 flowchart.

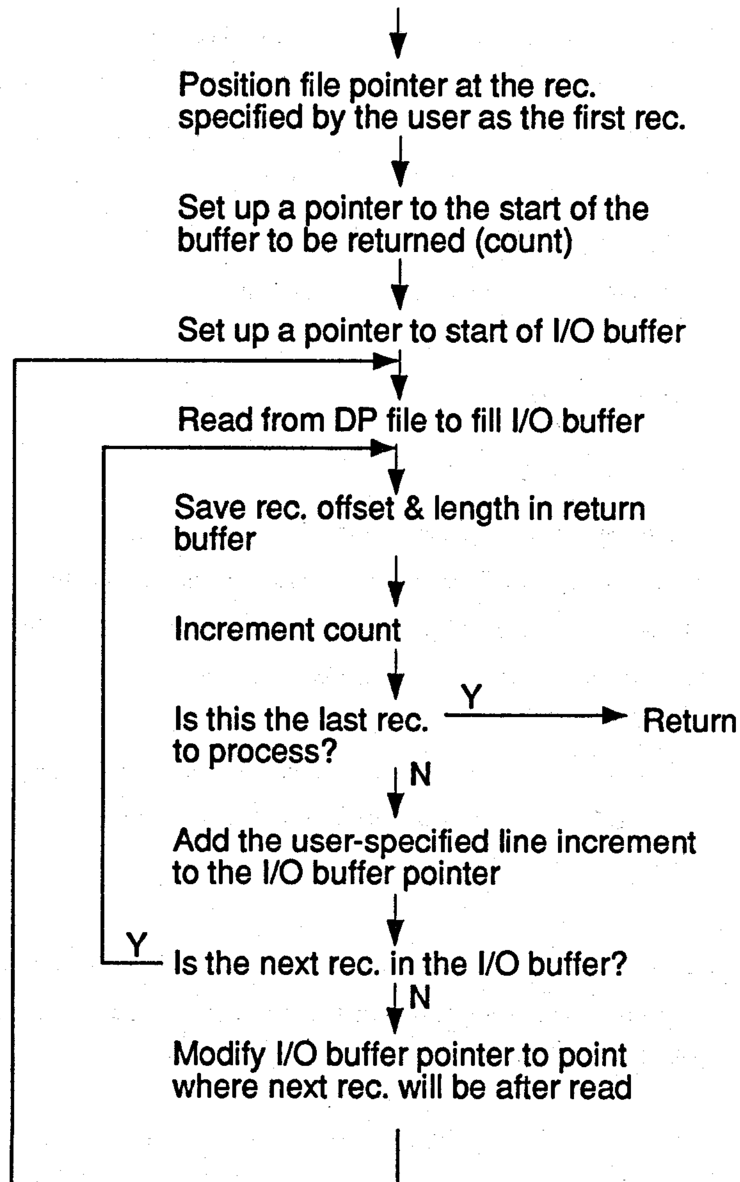

Figure 19. rec_sel_rec2 flowchart.

are known and the actual record numbers are unknown. Therefore, this function has to determine which records most closely match a set of target depths. See the Figure 20 flowchart. The function starts with the last specified depth as the first target depth. Three differences are calculated for each record processed:

$$\text{prev_dif} = |\text{depth of previous record} - \text{target depth}|$$
$$\text{cur_dif} = |\text{depth of current record} - \text{target depth}|$$
$$\text{next_dif} = |\text{depth of next record} - \text{target depth}|$$

The following condition must be met:

$$\text{prev_dif} > \text{cur_dif} <= \text{next_dif}$$

for the current record to become a selected record. Once a record has been selected, the target depth is decremented by the user-specified increment value and processing continues.

Since the cur_dif becomes the prev_dif and next_dif becomes cur_dif for the next record, only the next_dif needs to be calculated for each record processed.

12. disp_recs

    Using the list of selected records provided by one of the *_sel_rec functions, disp_recs reads and scales the data. The scaled data are then written to the specified location of a display channel. See Figure 21.

13. recs_specs

    If records were selected on the basis of depth, this function tells the user which records were selected as the first and last displayed. The corresponding depths of the first and last records specified are displayed if in record mode.

C. BATD User's Guide

1. Menu Description. BATD is a menu-driven algorithm that provides a flexible user interface. Movement within menus is accomplished using the arrow keys. A menu item is selected by pointing to an item with the arrow keys followed by *Enter* or by pressing the first character in the menu item name. To move up one level in the menu, use the Escape key.

Four options make up the upper level menu. The Source group includes all the parameters used to specify data to be displayed. The actual display location is controlled by means of options in the Display group. Run will display the specified data. Exit returns the user to the operating system.

An overview of the menu structure is shown in Figure 22. Each of the options grouped under Source and Display will be discussed in the following segments:

Source Filename

Specify both the data and DP files from which data are to be displayed.

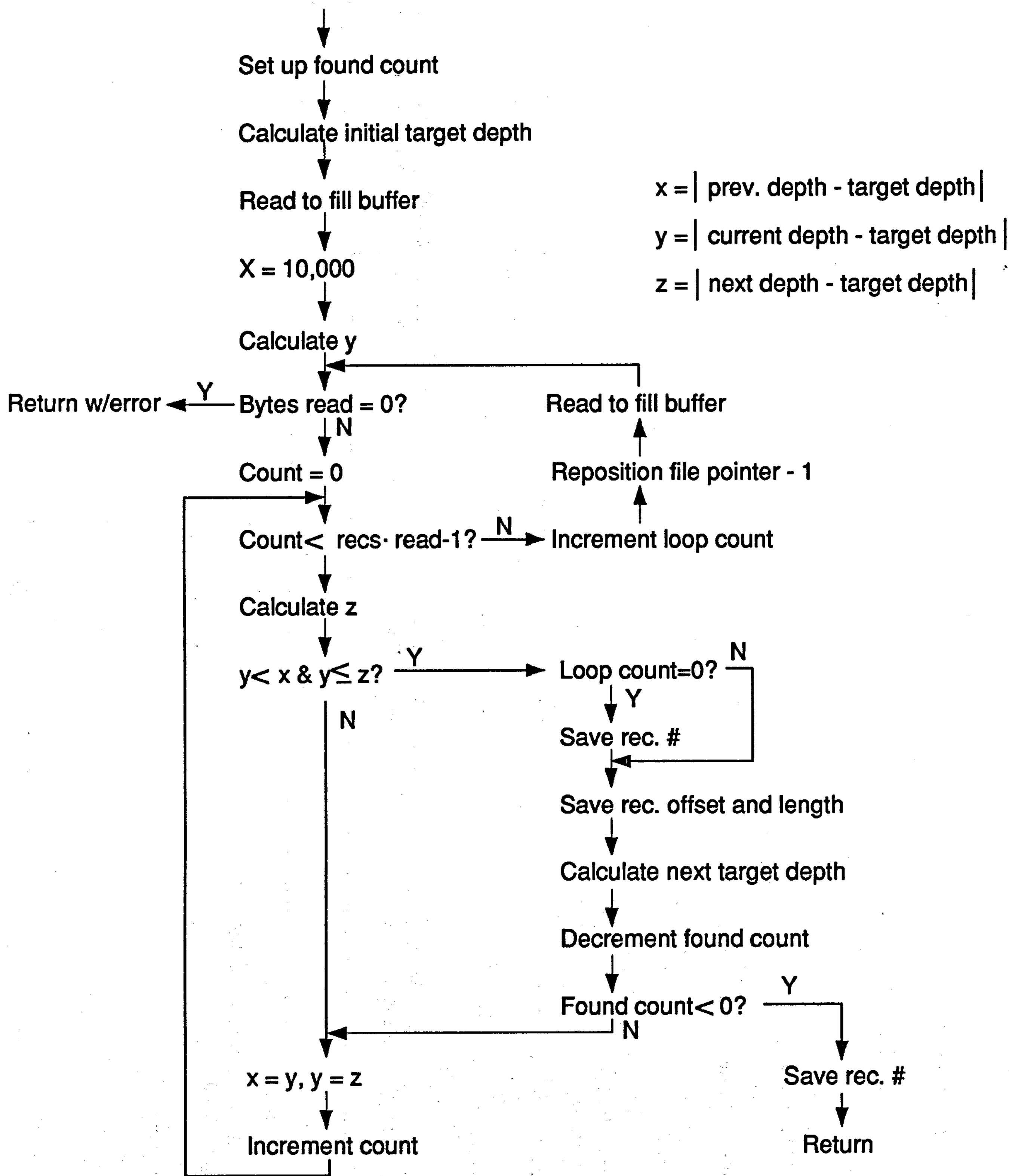

Figure 20. depth_sel_rec flowchart.

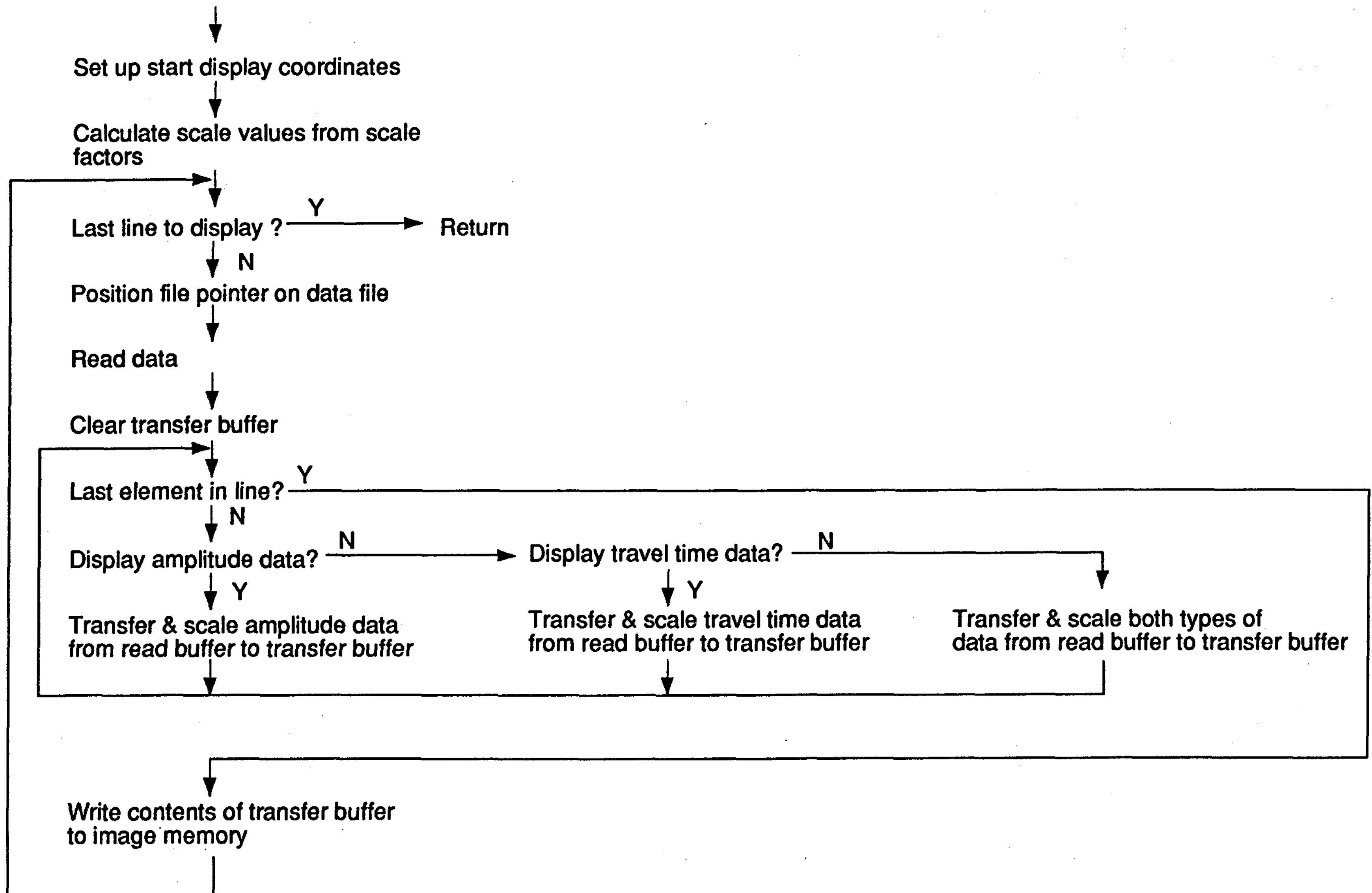

Figure 21. disp_recs flowchart.

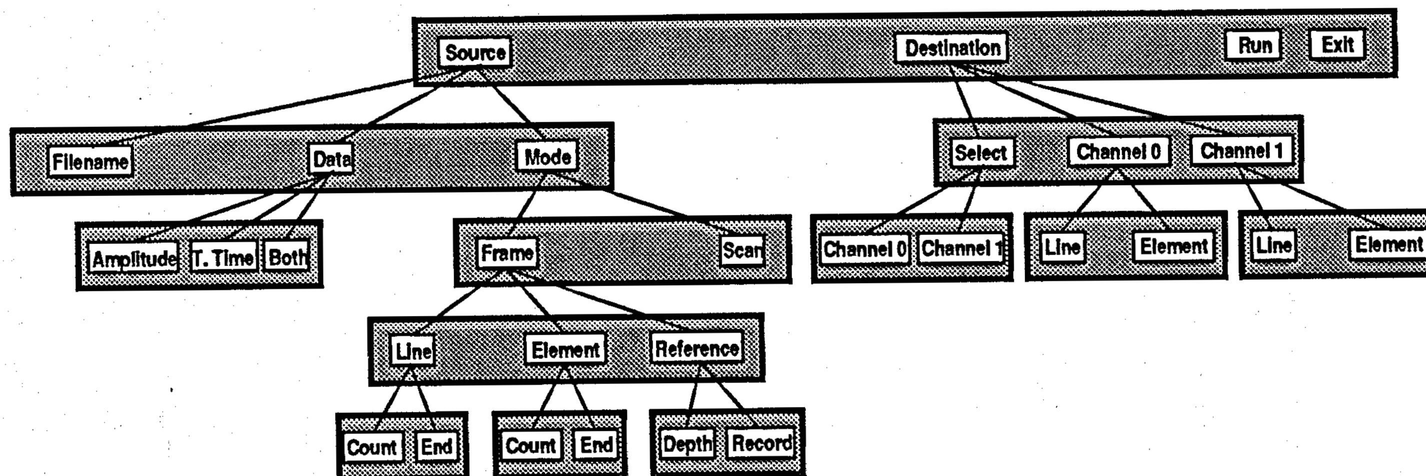

Figure 22. BATD menu structure.

Source Data Amplitude

Display only the amplitude data (see Source Data Both).

Source Data T. time

Display only the travel time data (see Source Data Both).

Source Data Both

Display the amplitude data followed by the travel time data. A single column of intensity 0 will be inserted to separate the two data types.

Selecting any Source Data option will result in a prompt for a set of scale values. For Source Data Amplitude or Source Data T. time, only the appropriate scale value is needed. Scaling of data is described in the following equation:

$$\text{display value} = \text{data value} / 2^{(\text{scale value})}$$

Scale values should be in the range of 0 to 7 inclusive.

Source Mode Frame Line Count

Source Mode Frame Line End

Data will be displayed as a single frame. A data form will ask which data records are to be displayed. These records may be specified by either a *count* or an *end* specification. The reference by which these records are specified is selected by Source Mode Frame Reference *. See Selecting Records Section below.

Note that the status window contains the first and last record numbers/depths contained in the file. This information is included to help generate the record specification. The record specification is checked extensively to make sure that the specified records are in fact contained within the data file.

Source Mode Frame Element Count

Source Mode Frame Element End

Identical to Source Mode Frame Line * except that elements from each data record are being selected.

Source Mode Frame Reference Depth

The records selected by Source Mode Frame Line * are specified by depth. A positive increment is required.

Source Mode Frame Reference Record

The records selected by Source Mode Frame Line * are specified by record number. The increment may be either positive or negative. See Logical Display Section below.

Source Mode Scan

The entire data file will be displayed by scrolling the display.

Scan mode is currently not implemented.

Destination Select Channel 0

Destination Select Channel 1

When Run is executed, the specified data will be written to the selected display channel. These options also select which channel is displayed. By using these options, unique data can be written to both channels.

Destination Channel 0 Line

Destination Channel 1 Line

Select the first display line to contain the first data record.

Display lines are numbered 0 - 479 beginning at the top of the display.

Destination Channel 0 Element

Destination Channel 1 Element

Select the first display column to contain the first data element.

Display columns are numbered 0 - 511 beginning at the left of the display.

When this menu structure is used, output similar to that shown in Figure 23 can be generated. Figure 24 contains the parameters used.

2. Selecting Records. Selecting records can be accomplished by either the *count* or the *end* methods. For either method, the first two parameters are a start and an increment value. The start value represents the first record to display. The increment value represents how many records to move from the first record to obtain the second and so on. For example, if the start value is 4 and the increment is 2, records 4, 6, 8,... would be selected.

The *count* or *end* specifications differ only in how the total number of records to select is determined. For *count*, the number of records is directly specified. In the *end* method, the approximate last record to select is specified by the user. The algorithm uses the start, increment, and end values to calculate the actual count. Then, using the start, increment, and calculated count, the algorithm determines the exact end value. The exact end value is used as the record specification and is written to the status window.

All values used in the record specification are limited to integer values.

3. Logical Display. As previously described, record selection may be made with respect to depth or record numbers. When depth reference is used, the display is limited to the *logical* format. In this format, the data from the shallowest portions of a wellbore are displayed at the top of the display, which provides a logical increase of depth as the eye travels down the display. For depth reference, the start depth must be smaller than the end depth and the increment must be positive.

The logical format restriction is removed when a record reference is used. This reference provides the user with additional freedom by supporting both positive and negative increments. Note that, when a positive increment is used in record reference, the data are not displayed in *logical* order. Only with a negative increment can this format be duplicated.

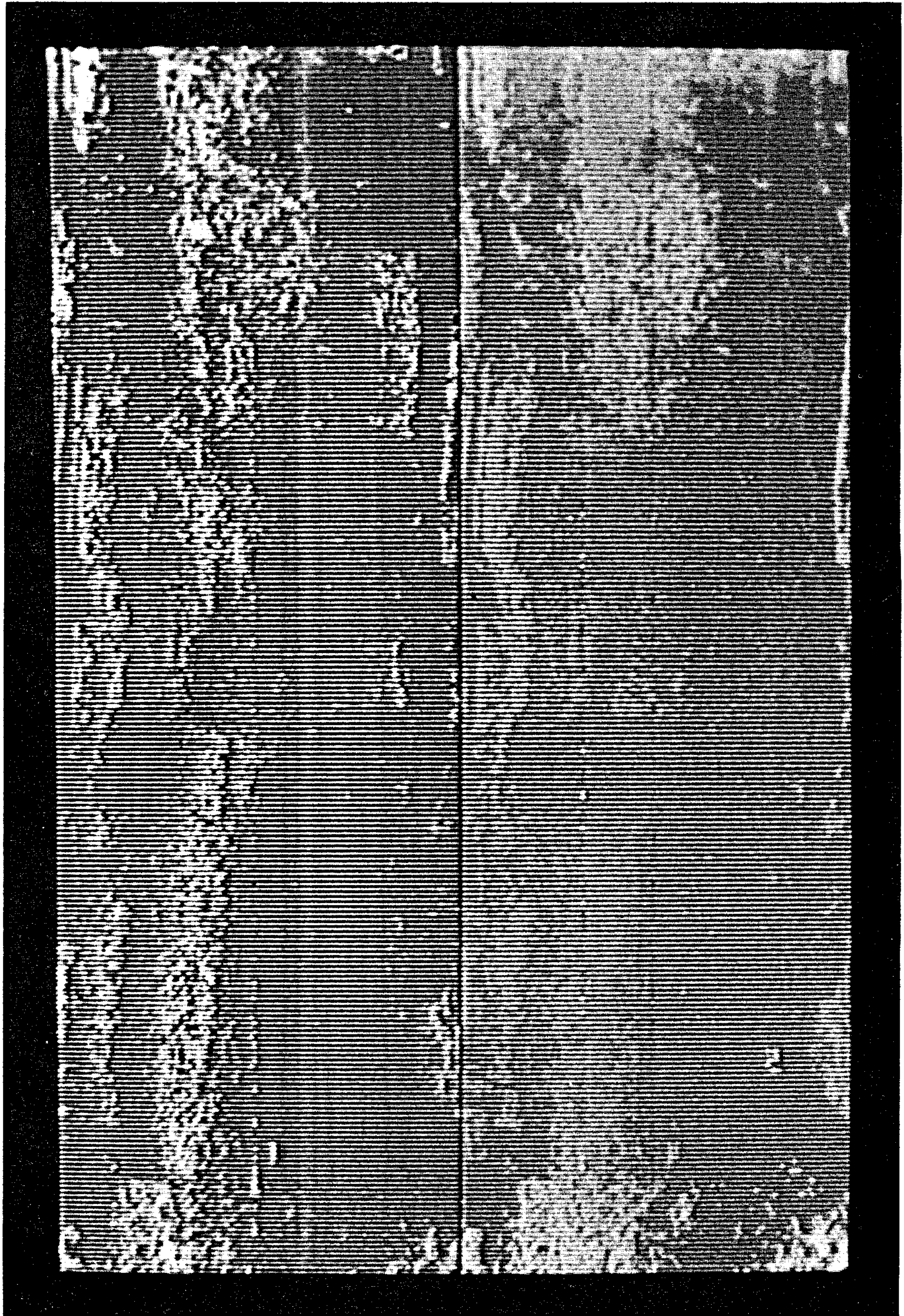

Figure 23. Example of display generated by BATD.

```
**************************************************
*     Starting record    1100 -> Depth  8667.5     *
*     Final record        621 -> Depth  8682.0     *
************ Strike any key to continue ***********
```

```
***************************************************************************
* Source                                                                  *
*  Data Fname.087085t3.raw           DP Fname...087085t3.dpf              *
*  Recs..First: 1  Last:   4863      Depth.First:  8700.9 Last:  8553.3   *
*  Data type..Both                   Scale Factor....Amp:  2    TT:  4    *
*  Mode.......Frame  Ref...Record   Destination                           *
*          Start Count Inc  End      Selected Channel......Channel 0      *
*    Line   1100   480  -1   621     Channel 0....Line:   1   Elem:  128  *
*    Elem      1   128   1   128     Channel 1....Line:   1   Elem:    1  *
****************************** Status Window ******************************
```

Figure 24. Parameters used to generate Figure 23.

VIII. BATT (BAT Trace Generation Utility)

A. Functional Description

BATT is a utility designed to plot the travel time data as radial distance in inches vs sensor rotational position. Each plot generated on the display, referred to as a trace, is from a single revolution. Records from which to generate traces are identified with reference to depth by the user. Lines of reference, referred to as tick lines, and scaling parameters are specified by the user. The minimum and maximum radii found in the records in the interval bounded by two successive trace records are added to the display. Each trace is annotated with depth and tick line reference information. A video copier can easily generate hardcopies.

This utility is menu driven with pull-down submenus to provide the user with a friendly interface. Data forms are used to obtain parameters from the user. A status window is constantly updated to provide the user with current parameters. Windows relay additional information or error messages to the user.

B. Implementation

BATT is a complex algorithm simplified by the extensive use of functions. The main routine is predominantly a collection of function calls, which break the complex algorithm up into more understandable, smaller tasks. Figure 25 provides an overview of the algorithm.

Two special libraries of functions were used to help develop this code. The first, *Windows for Data*, Vermont Creative Software, was used to design/implement the menu structure. This library was also used to provide informational windows as well as data forms for entering data. Access to the image processing boards was provided by the DT-IRIS library from Data Translation.

To avoid passing numerous parameters between functions, global variables are used extensively. Information pertaining to these variables as well as to all the *#defines* is contained in a set of include files. See Appendix B.

In developing BATT, it was assumed that the typical user would, upon selecting the records to use, spend most of his/her energies on fine tuning the display parameters. Since it takes much longer to identify the records and generate the interval calculations, all parameters selected or changed are watched closely. Only when the user changes a record selection parameter is the entire algorithm executed. This results in much faster execution when the user is simply adjusting display parameters.

To provide insight into the concepts involved in the implementation of this algorithm, each of the functions developed for BATT will be discussed in the following pages. Flowcharts are provided for the more involved functions.

1. init_parms

Initializes many of the global parameters to default values. Also sets up the column indices of the display coordinates for the tick lines and traces.

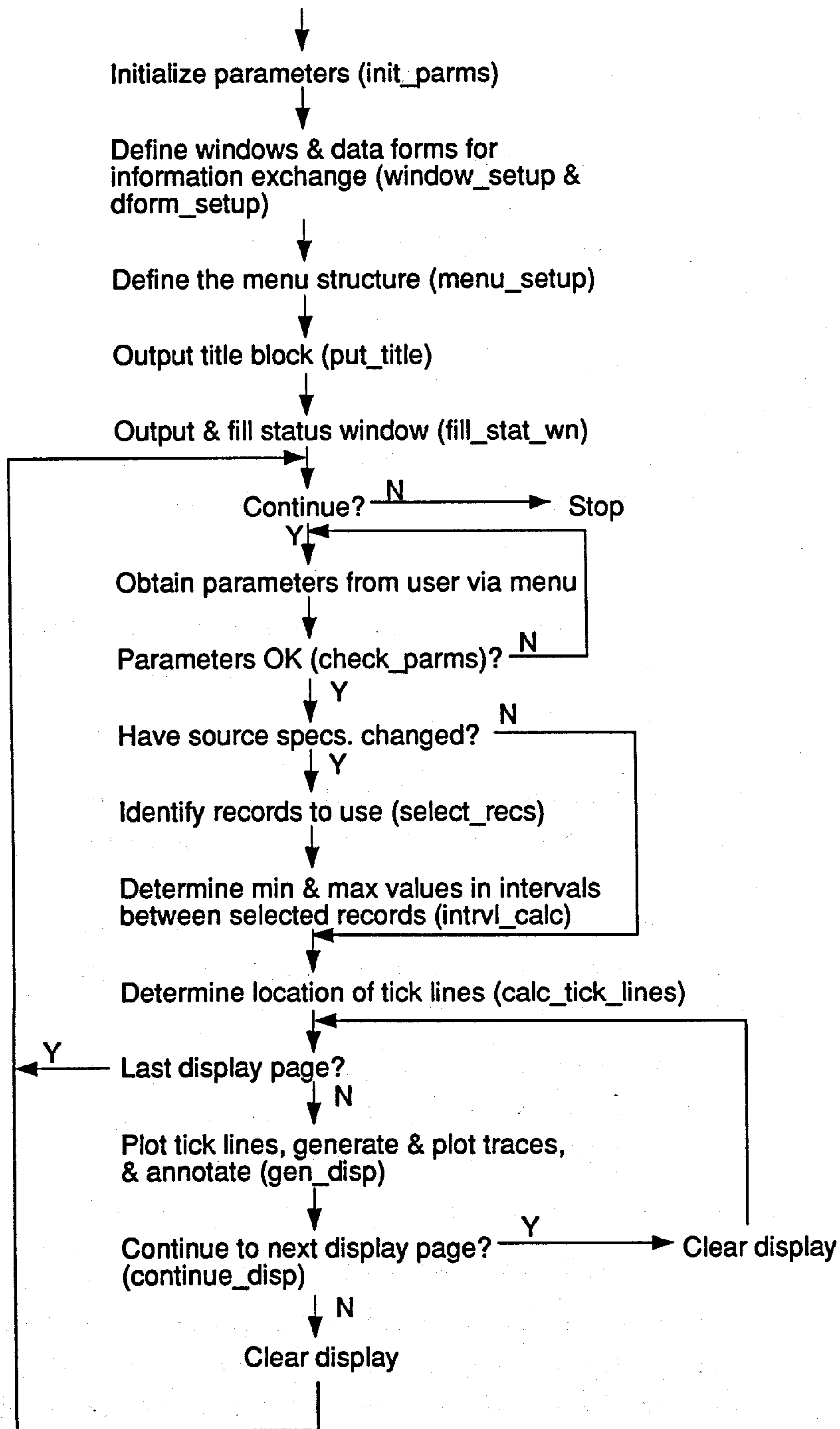

Figure 25. BATT flowchart.

2. window_setup

Defines the windows used to provide information to the user.

3. dform_setup

Defines the data forms used to obtain parameters from the user.

4. menu_setup

Defines the menu structure including those action functions called as the result of a menu item being selected.

5. put_title

Places the title window on the console at startup.

6. fill_stat_wind

At startup, fills the status window with parameter values by repeated calls to the function stat_wind_wrt. stat_wind_wrt updates a specific line in the status window and is called any time a parameter changes.

7. Action functions

When a bottom-level menu option is selected, one of five action functions will be called. These functions will result in obtaining parameter(s) from the user, exiting the menu structure to generate traces, or exiting the algorithm.

8. check_parms

Checks all user-entered parameters before the algorithm is permitted to continue. If any invalid parameters are located, the user is notified and returned to the menu structure.

9. select_recs

This function determines which records most closely match a set of target depths. See the Figure 26 flowchart. The function starts with the last specified depth as the first target depth. Three differences are calculated for each record processed:

prev_dif = |depth of previous record - target depth|
cur_dif = |depth of current record - target depth|
next_dif = |depth of next record - target depth|

The following condition must be met:

prev_dif > cur_dif <= next_dif

for the current record to become a selected record. Once a record has been selected, the target depth is decremented by the user-specified increment value and processing continues.
Since the cur_dif becomes the prev_dif and next_dif becomes cur_dif for the next record, only the next_dif needs to be calculated for each record processed.

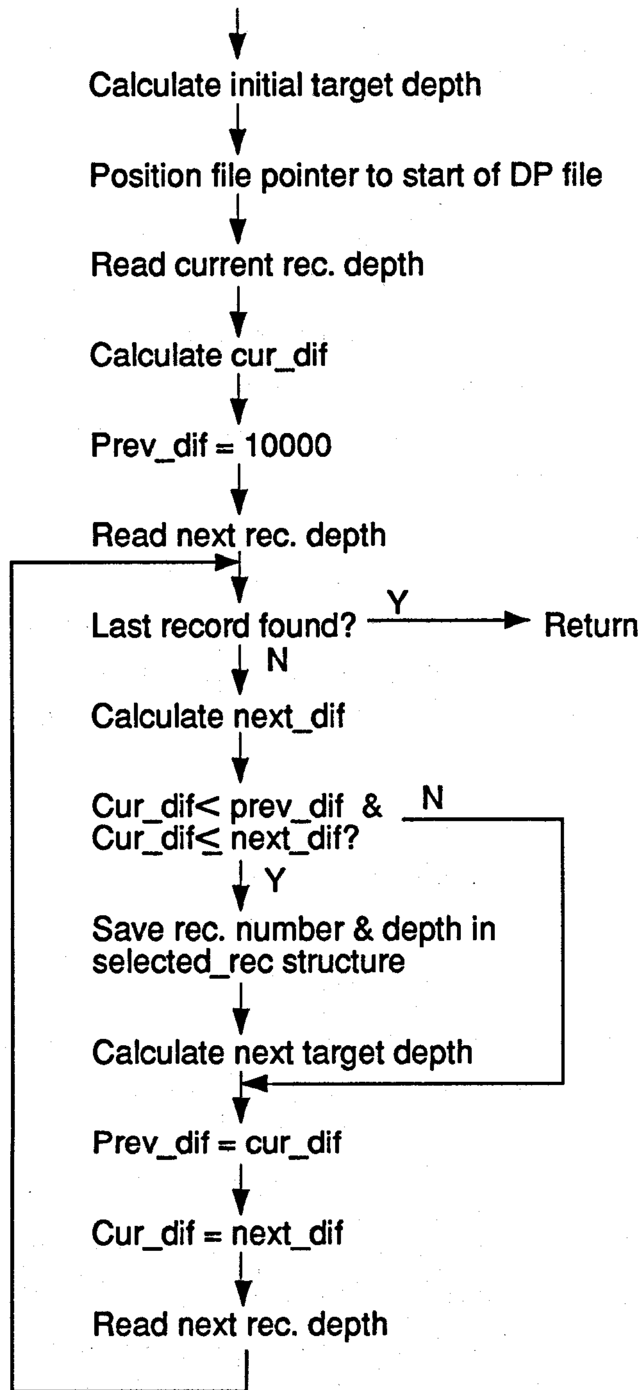

Figure 26. select_recs flowchart

select_recs collects the depth and record number of each record identified in the selection process outlined above. This information is placed in the data structure named selected_recs for use by subsequent functions.

10. intrvl_calc

Determines the minimum and maximum radii found in the interval bounded by successive records identified by select_recs. The two bounding traces are not included in this calculation.

The resulting minimum and maximum are added to the selected_rec data structure partially filled in by select_recs. For a given interval, the minimum and maximum values are placed with the shallower bounding selected record. The final selected record therefore has no valid minimum and maximum values. See Figure 27.

11. calc_tick_lines

Calculates where the reference tick lines should be drawn. Since the starting and ending column are fixed by the number of travel time data points per record, only the line number is actually calculated. The number of traces per display page determines tick line location. See the Figure 28 flowchart. The results of this function are critical as gen_disp generates the traces with respect to the tick lines.

12. gen_disp

gen_disp is a busy function that handles three tasks:

a) draws the tick lines,

b) maps and draws the actual traces, and

c) adds all alphanumeric annotation.

The selected records, as specified in data structure selected_recs, must be located and read. gen_disp translates the record number into an offset into the data file. The travel time is read into a buffer and processed a word at a time. Each data point is mapped into physical display coordinates in three logical steps:

a) convert from caliper in millimeters X 10 to radius in inches,

b) calculate the offset from a base tick line in display lines, and

c) identify the physical line number with respect to the base tick line.

An array of display coordinate pairs is generated from the record. Once all the data from a single revolution are processed, the trace is drawn. See Figure 29.

13. continue_disp

Determines if the user wants to continue displaying traces or return to the menu.

C. BATT User's Guide

1. Menu Description. BATT is a menu-driven algorithm that provides a flexible user interface. Movement within menus is accomplished using the arrow keys. A menu item is selected by pointing to an item with the arrow

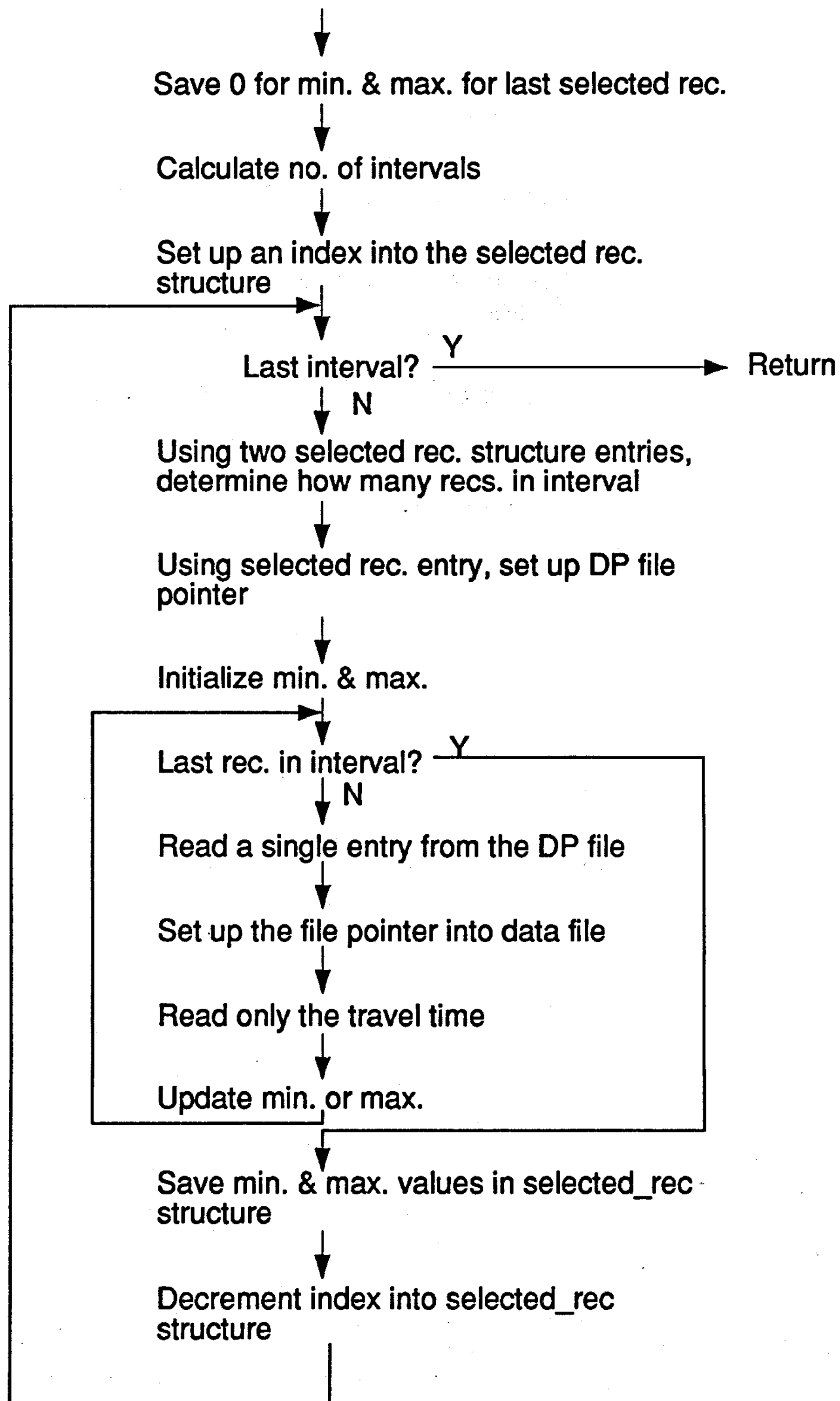

Figure 27. intrvl_calc flowchart.

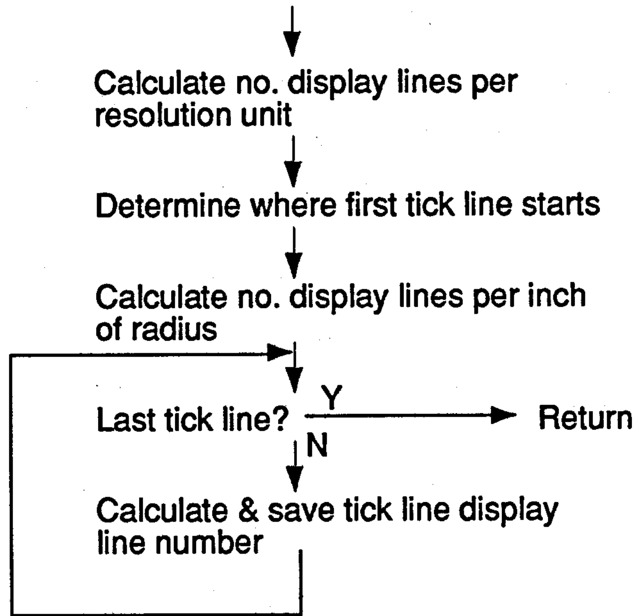

Figure 28. calc_tick_lines flowchart.

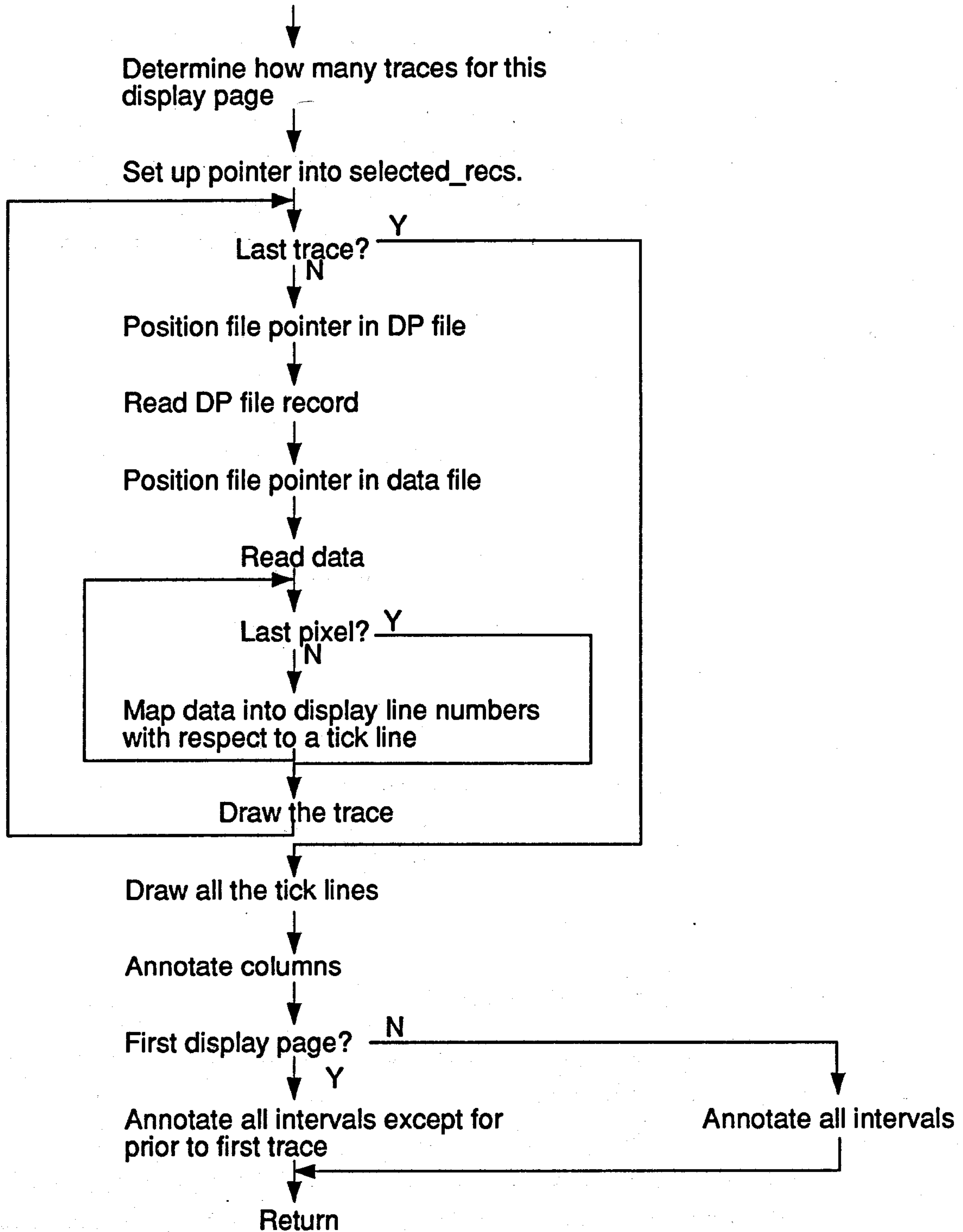

Figure 29. gen_disp flowchart.

keys followed by *Enter* or by pressing the first character in the menu item name. To move up one level in the menu, use the Escape key.

Three options make up the upper level menu. The Parameters group includes all the parameters used to specify data from which traces are to be generated and the way in which those traces are to be displayed. Display will draw the specified traces. Exit returns the user to the operating system.

An overview of the menu structure is shown in Figure 30. Each of the options grouped under Parameters will be discussed in the following segments.

Parameters Filenames

Specify both the data and the DP files from which traces will be generated.

Parameters Select Recs

Specify records with respect to depth from which to generate traces using the *end* method. See Section VII.C.2 for record specification details.

Parameters Display

User determines how the traces are to be placed on the display. The number of traces per display, base tick line value, and resolution between tick lines can be specified by this menu selection. See the next segment for definitions.

These menu items were used to generate the traces shown in Figure 31. Parameters used are included in Figure 32.

2. Display Definition. Several concepts need to be discussed in order to understand what the displayed traces represent and how the parameters under Parameters Display can be used. Consider the example shown in Figure 33. To generate this example, the following display parameters were used:

Traces per display:	2
Base tick line:	4.0 in.
Resolution between tick lines:	1.0 in.

The number of traces per display controls the maximum number of traces that can be shown on a single display. If more traces have been specified by means of Parameters Select Recs, several successive displays, referred to as pages, are generated.

Each trace is generated with respect to a base tick line. The value in inches that the base tick line represents is specified as the base tick line value. In Figure 33, line b is the base tick line for Trace 1. Trace 2's base tick line is c.

In addition to a base line, some unit of scale must be added to the display. The resolution-between-tick-lines parameter controls the scale. This parameter specifies the physical distance represented on the display by the distance between two successive tick lines. This resolution applies to all the tick lines on the display. To illustrate this last point, Table I defines what each tick line represents to each trace.

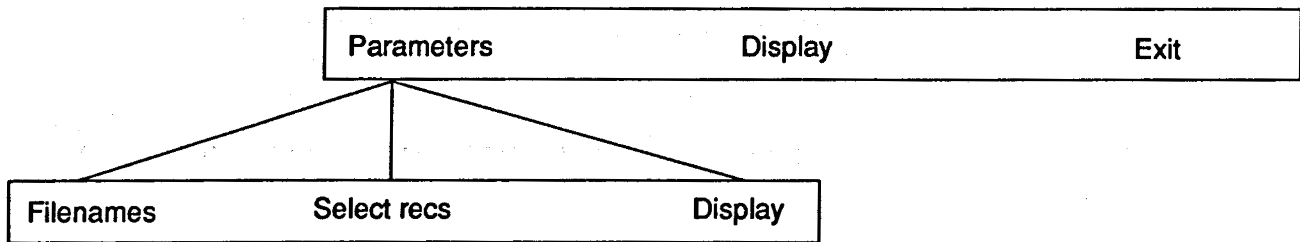

Figure 30. BATT menu structure.

Figure 31. Typical traces generated by BATT.

```
****************************************************
*   Page 1 of 3 total pages                        *
*   Enter Ctrl-C to return to the menu structure   *
************ Strike any key to continue ***********
```

```
****************************************************************************
* Source                                                                   *
*  Data Fname: tape1.raw                                                   *
*  DP Fname:   tape1.dpf                                                   *
*  Min depth in file:   900.03        Max depth in file:  1000.96          *
*  Start depth:     910.00            End depth:     920.00                *
*  Increment:         1.00            No of traces:   11                   *
*                                                                          *
*  Display                                                                 *
*  Total number of displays:   3      Number traces per display:   4       *
*  Base tick line:    4.00            Resolution between tick lines:  1.00 *
****************************** Status Window *******************************
```

Figure 32. Parameters used to generate Figure 31.

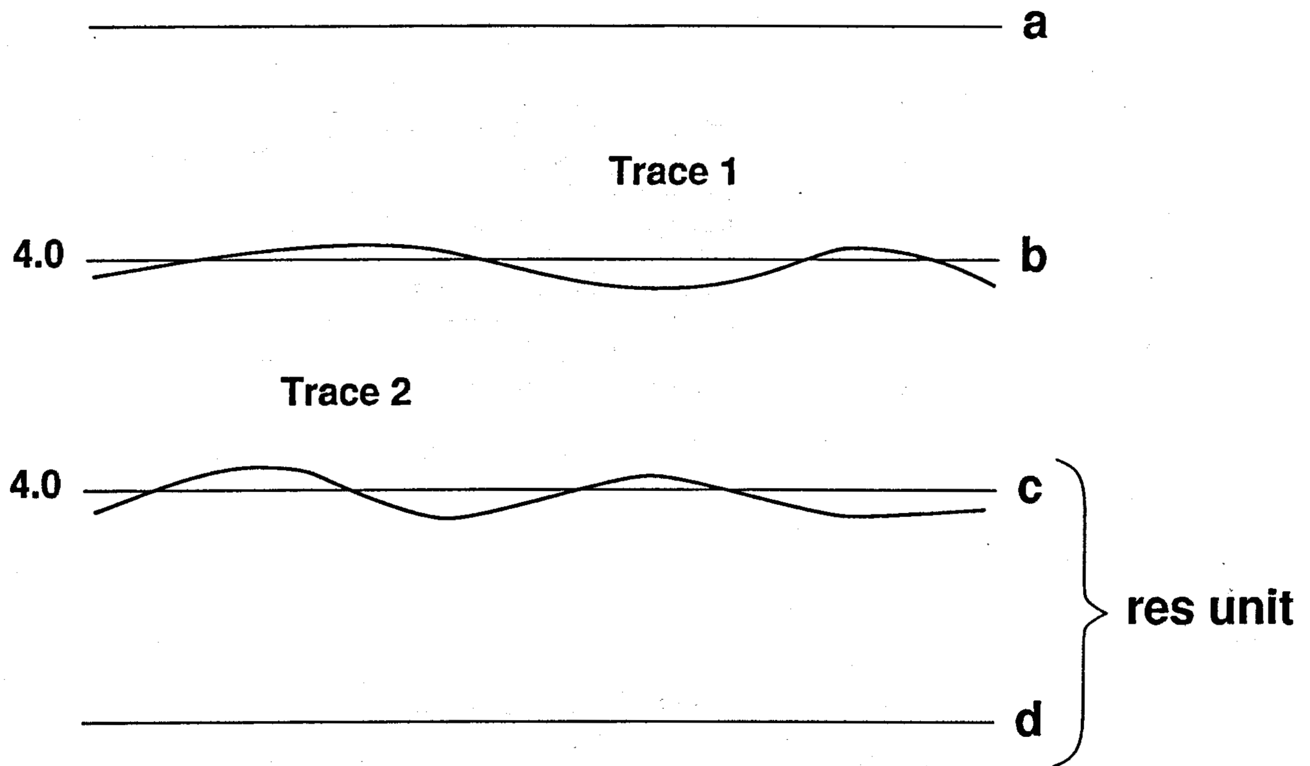

Figure 33. Example of two traces with associated tick lines.

TABLE I
TICK LINE REPRESENTATIONS

With Respect to Trace	Tick Line	Radius of
1	a	Base tick line + 1 res unit = 4 + 1 = 5 in.
1	b	Base tick line = 4 in.
1	c	Base tick line - 1 res unit = 4 - 1 = 3 in.
1	d	Base tick line - 2 res units = 4 - 2 = 2 in.
2	a	Base tick line + 2 res units = 4 + 2 = 6 in.
2	b	Base tick line + 1 res unit = 4 + 1 = 5 in.
2	c	Base tick line = 4 in.
2	d	Base tick line - 1 res unit = 4 - 1 = 3 in.

Notes:

1) The base line for Trace 1 was tick line b, and for Trace 2 it was tick line c.

2) A specific tick line represents different radii for different traces on the same display.

3) By definition, each trace generated will always have at least three tick lines by which it can be referenced. They are a base tick line, base tick line + 1 res unit, and base tick line - res unit.

3. Usage Notes. Much flexibility is provided in generating the trace display, but the user should be careful. For example, a base tick line value that is too big or too small coupled with a small value for the resolution between tick lines will produce traces that may be several resolution units away from the base tick lines. On the other hand, too large a resolution value will result in very little detail in the resulting trace.

The maximum number of traces per display that will still provide readable annotation is 32.

REFERENCES

1. K. Hinz and R. Schepers, 1981, "SABIS--The Digital Version of the Borehole Televiewer," WBK, Institut fur Geophysik, Bochum, FDR.

2. T. K. Moore, K. Hinz, and J. Archuleta, "Development of a New Borehole Acoustic Televiewer for Geothermal Applications," *1985 International Symposium on Geothermal Energy*, Vol. II (Geothermal Resources Council, Davis, California, 1985).

3. Vermont Creative Software, *Windows for Data Reference Manual*, Version 2.06.

4. Vermont Creative Software, *Windows for C Reference Manual*, Version 4.14.

5. Data Translation, Inc., *DT-IRIS Reference Manual*, Version 1.00.

Appendix A - Workstation Development Environment

Hardware

IBM PC AT (6 MHz)
Core AT260 Disk Unit
Core ESDI Disk Controller
IBM 3.5-in. Disk Controller and External Drive Unit
AST Advantage Multifunction Board
IBM EGA Monitor
Video 7 Vega Deluxe EGA Adapter
Microsoft Bus Mouse
Maynard Maynstream Interface
Data Translation DT 2851 High Resolution Frame Grabber
Data Translation DT 2858 Auxiliary Frame Processor

Software

DOS 3.30 Operating System
Microsoft C Compiler, V4.0
Brief Programmers Editor,* V2.0
Windows for C,** V4.14
Windows for Data,** V2.06
DT-IRIS Subroutine Library,*** V1.01

----

* Solution Systems.
** Vermont Creative Software.
*** Data Translation.

Appendix B - Disk Files Used to Generate Algorithms

Utility	File	Contents\Use
BDCOMP	\CODE\MSC\BAT\BDCOMP.C	All source code
	\CODE\MSC\BAT\BDCOMP.	Make file
DATASEP	\CODE\MSC\BAT\DATASEP.C	All source code
	\CODE\MSC\BAT\DATASEP.	Make file
DPDUMP	\CODE\MSC\BAT\DPDUMP.C	All source code
	\CODE\MSC\BAT\DPDUMP.	Make file
BDINVRT	\CODE\MSC\BAT\BDINVRT.C	All source code
	\CODE\MSC\BAT\BDINVRT.	Make file
BDEXTRCT	\CODE\MSC\BAT\BDEXTRCT.C	All source code
	\CODE\MSC\BAT\BDEXTRCT.	Make file
BATD	\CODE\MSC\BAT\BATD.C	Main function only
	\CODE\MSC\BAT\BATDACTS.C	All action functions
	\CODE\MSC\BAT\BATDRSEL.C	Functions depth_sel_rec, rec_sel_rec1, rec_sel_rec2, and disp_recs
	\CODE\MSC\BAT\BATDSTUP.C	Functions init_parms and menu_setup
	\CODE\MSC\BAT\BATDWIND.C	Functions window_setup put_title, fill_stat_wn, stat_wind_wrt, check_parms, error_handler, and recs_specs
	\COMP\MSC\INCLUDE\BATDGLOB.H	Global variable definitions
	\COMP\MSC\INCLUDE\BATDWND.H	Global window/data form definitions
	\COMP\MSC\INCLUDE\BATDXTRN.H	External variable declarations
	\COMP\MSC\INCLUDE\BATDXWND.H	External window/data form declarations
	\COMP\MSC\INCLUDE\BATDDEFS.H	All #defines
	\CODE\MSC\BAT\BATD.	Make file
	\CODE\MSC\BAT\BATD.CMD	Link command line
BATT	\CODE\MSC\BAT\BATT.C	Main function only
	\CODE\MSC\BAT\BATTSTUP.C	Functions init_parms, window_setup, dform_setup, and menu_setup
	\CODE\MSC\BAT\BATTPROC.C	Functions check_parms, select_recs, intrvl_calc, calc_tick_lines, gen_disp, and doub_convrt
	\CODE\MSC\BAT\BATTWIND.C	Functions put_title, fill_stat_wn, stat_wind_wrt, all action functions, error_handler, and continue_disp
	\COMP\MSC\INCLUDE\BATTGLOB.H	Global variable definitions
	\COMP\MSC\INCLUDE\BATTWND.H	Global window/data form definitions
	\COMP\MSC\INCLUDE\BATTXTRN.H	External variable declarations
	\COMP\MSC\INCLUDE\BATTXWND.H	External window/data form declarations
	\COMP\MSC\INCLUDE\BATTDEFS.H	All #defines
	\CODE\MSC\BAT\BATT.	Make file

Appendix C - Field Tape Format

Log Header Format*

Byte Offset	Length	Field Description
0 - 15	16	Identification string *Sabis Field Tape*
16 - 17	2	Tape number
18 - 23	6	Date log began in YYMMDD format
24 - 27	4	Time log began in HHMM format
28 - 29	2	Experiment number
30	1	Metric or English unit indicator
31	1	Mud log available (Y or N)
32 - 47	16	Well number/name
48 - 63	16	Location/field name
64 - 79	16	Country
80 - 95	16	State
96 - 101	6	Tool name
102 - 105	4	Transducer frequency in KHz
106 - 109	4	Shots per revolution
110 - 111	2	Revolutions per second
112 - 115	4	Tool software version
116 - 121	6	Control unit name
122 - 125	4	Control unit software version
126 - 127	2	Revolutions per revolution record
128 - 131	4	Number bytes in revolution header
132 - 135	4	Number amplitude data bytes
136 - 139	4	Number caliper data bytes
140 - 171	32	Logging company name
172 - 187	16	Logging engineer
188 - 219	32	Client company
220 - 235	16	Witness
236 - 251	16	Borehole fluid
252 - 267	16	Borehole fluid additives
268 - 271	4	Specific weight in g/l
272 - 275	4	Viscosity
276 - 279	4	Acoustic speed in km/s
280 - 287	8	Casing shoe depth in mm
288 - 295	8	Fluid depth in mm
296 - 303	8	Total depth in mm
304 - 307	4	Drilled caliper in mm
308 - 315	8	
316 - 323	8	
324 - 331	8	
332 - 511	179	Unused, filled with *0*
512 - 1023	512	Additional comment text

* All information is ASCII.

Appendix C - Field Tape Format

Revolution Record Format

Byte Offset	Length	Field Description
0 - 7	8	Identification string *Sabislog*
8 - 15	8	Collection depth
16 - 21	6	Collection time in HHMMSS format
22 - 25	4	Revolution counter
26 - 27	2	North indicator
28 - 29	2	Caliper offset
30	1	Fire voltage
31	1	Amplitude attenuation
32 - 33	2	Temperature #1
34 - 35	2	Temperature #2
36 - 37	2	Z-axis accelerometer minimum
38 - 39	2	Z-axis accelerometer maximum
40 - 41	2	Y-axis tilt
42 - 43	2	X-axis tilt
44 - 45	2	Flux gate intensity
46	1	Interest indicator
47	1	Unused
48 - 303	256	128 words of amplitude data
304 - 559	256	128 words of travel time data

Appendix D - Typical Terminal Session Output

```
C:>bdcomp

BAT Tool Data Complement and Header Fix Algorithm
               V1.0, Troy K. Moore

Enter input filename:   tape1.rtd
Enter output filename:  tape1.ctd
Possible fix at end of file ignored

2 fixups were made to log header records
18 repositions were executed
2097152 bytes were processed

C:>datasep

*************************************************************
*          BAT Tool Data Separation Algorithm               *
*         ESS-4, Los Alamos National Laboratory             *
*************************************************************
*               Developed by Troy K. Moore                  *
*             Version: 1.11  Dated: 11/9/87                 *
*************************************************************

Enter input file name:  tape1.ctd

Default output file names are as follows:
         Data file:  TAPE1.RAW
         Data pointer file:  TAPE1.DPF
         Log header file:  TAPE1.LHF

To accept these file names, hit the Enter key.
To generate alternate file names, hit any other key.

*************************************************************
*                 Data Separation Complete                  *
*************************************************************
*     Revolution data records processed:     3741           *
*          Log header records processed:        2           *
*************************************************************
```

Appendix D - Typical Terminal Session Output

```
C:>dpdump tape1.dpf 560 10

Data Pointer File Dump Utility, Version 1.0

File name: tape1.dpf                                         Flag: 560 +/-10

Record   Offset                 Record | Record   Offset              Record
Number   in hex      Depth      Length | Number   in hex      Depth   Length
-----------------------------------+---------------------------------
     1         0   1000.96      560   |       57      7a80   1000.96     560
     2       230   1000.96      560   |       58      7cb0   1000.96     560
     3       460   1000.96      560   |       59      7ee0   1000.96     560
     4       690   1000.96      560   |       60      8110   1000.96     560
     .         .         .        .           .         .         .       .
     .         .         .        .           .         .         .       .
     .         .         .        .           .         .         .       .

C:>bdinvrt

BAT Data Inversion Utility
Developed by Troy K. Moore
    Version 1.00

<Input Filenames>
     DP File:    tape1.dpf
     Data File:  tape1.raw

<Output Filenames>
     DP File:    tape1in.dpf
     Data File:  tape1in.raw

3741 records were processed

C:>bdextrct

BAT Data Extraction Utility
Developed by Troy K. Moore
    Version 1.00

<Input Filenames>
     DP File:    tape1.dpf
     Data File:  tape1.raw

<Output Filenames>
     DP File:    tape1ex.dpf
     Data File:  tape1ex.raw

<Record Specifications>
     Start extraction with record:  45
     Final record to extract:       90

50 records were extracted

C:>
```

Appendix E - BDCOMP Source Code

```
/*
*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*
•
*   MODULE NAME:  bdcomp
•
*   DESCRIPTION:  This algorithm is designed to process raw BAT tool data
*     transferred from field tapes into the expected format.  This includes
•
•        - all data, except for the next case, are complemented
•        - the 'S' in Sabis Field Tape is replaced
*
•     This algrothim is designed to be executed with command line arguments
*     or interactively.
•
•   NOTES:  The deletion of the 'S' is a problem with early versions of WBK's
*     uphole software.  This problem will probably be corrected in the near
*     future.  This code will work correctly in either case.
•
*     The test for 'S' substitution is that the candidate character is 0xff
•     and that the following 4 characters are the complement of 'abis'.
*     The only foolproof test would be to match all the remaining characters
•     of the keystring ('abis Field Tape').  It was felt that to match the
•     four characters was a sufficient test.
•
*     The handling of command line parameters is different with pre 3.0
•     versions of DOS.
•
•     Command line syntax:    bdcomp input_file output_file
*
•     A good test for the repositions for possible fixes occurs when the
*     buffer length is set to 1066 and TAPE2.DAT is used as input.  Numerous
•     repositions occur but the resulting output file is the same as with
*     a 24 Kbyte buffer.
•
*   DEVELOPED BY: Troy K. Moore
•   DATE:         9/25/87
•
•   REVISION NOTES:
*
*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*
*/

#include    <stdio.h>
#include    <fcntl.h>
#include    <sys\types.h>
#include    <sys\stat.h>

#define     BUF_LEN      24576
#define     FNAME_LEN    60

main(argc,argv)

int   argc;
char  *argv[];
```

```
{
   static char     in_fname[FNAME_LEN], out_fname[FNAME_LEN];
   static char     io_buf[BUF_LEN];
   char            *point1, *point2;

   int             in_fhand, out_fhand, count, fix_count;
   int             bytes_read, bytes_write, repos_count;

   long int        total_bytes, sflag, soffset;

   long int        lseek();


/* Print out header info */

   printf("\nBAT Tool Data Complement and Header Fix Algorithm\n");
   printf("               V1.0, Troy K. Moore\n\n");

/* Prompt user for file names if not passed via command line */

   if (argc != 3)                      /* may be different on pre 3.0 DOS */
   {
      printf("Enter input filename:   ");
      scanf("%s",in_fname);
      printf("Enter output filename:  ");
      scanf("%s",out_fname);
      point1 = in_fname;               /* initialize pointers to strings */
      point2 = out_fname;
   }

/* Initialize pointers to passed strings */

   else
   {
      point1 = argv[1];
      point2 = argv[2];
   }

/* Open input file */

   in_fhand = open(point1,O_RDONLY|O_BINARY);
   if (in_fhand == -1)
   {
      perror("\nError opening input file");
      exit(-1);
   }
```

Appendix E - BDCOMP Source Code

```
/* Create output file */

   out_fhand = open(point2,O_CREAT|O_BINARY|O_RDWR,S_IREAD|S_IWRITE);
   if (out_fhand == -1)
   {
      perror("\nError creating output file");
      exit(-1);
   }

/* Prepare for main loop */

   fix_count = 0;
   repos_count = 0;
   total_bytes = 0L;
   bytes_read = read(in_fhand,io_buf,BUF_LEN);

/* Start main control loop */

   while (bytes_read > 0)
   {

/* Process a buffer full of data making sure that any partial matches
   of the log header keystring occur within the boundaries of the buffer */

      for(count = 0; count < bytes_read; count++, total_bytes++)
      {
         if (io_buf[count] == (char)0xff)
         {
            if (count < bytes_read - 4)   /* match before end of buffer? */
            {
               if (io_buf[count+1] == ~('a') && io_buf[count+2] == ~('b') &&
                  io_buf[count+3] == ~('l') && io_buf[count+4] == ~('s'))
               {
                  fix_count++;            /* partial keystring matched, subs */
                  io_buf[count] = 'S';
               }

               else                       /* no partial match after 0xff */
                  io_buf[count] = ~io_buf[count];

            }

            else              /* too close to the end of buffer for match */
            {
               if (bytes_read == BUF_LEN) /* if true, not at end of file */
               {
                  soffset = (long int)(count - bytes_read);
                  sflag = lseek(in_fhand,soffset,SEEK_CUR);
                  if (sflag == -1L)
                  {
                     perror("\nError repositioning file pointer");
                     exit(-1);
                  }
```

```
                repos_count++;
                total_bytes--;          /* otherwise byte is counted 2x */
                bytes_read = count;     /* adjust for upcoming write */
             }

             else                       /* at end of file, pass char as is */
                printf("Possible fix at end of file ignored\n");

          }

       }

       else                             /* typical character processing */
          io_buf[count] = ~io_buf[count];

    }

    bytes_write = write(out_fhand,io_buf,bytes_read);
    if (bytes_write != bytes_read)
    {
       perror("\nError writing to output file");
       exit(-1);
    }

    bytes_read = read(in_fhand,io_buf,BUF_LEN);

 }

/* Handle any errors detected during reads */

 if (bytes_read == -1)
 {
    perror("\nError reading from input file");
    exit(-1);
 }

/* Output processing statistics */

 printf("\n%d fixups were made to log header records\n",fix_count);
 if (repos_count != 0)
    printf("%d repositions were executed\n",repos_count);
 printf("%ld bytes were processed\n",total_bytes);

/* Terminate execution */

 close(in_fhand);
 close(out_fhand);

 exit(0);

}
```

Appendix F - DATASEP Source Code

```
/*
*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*
*
*   MODULE NAME:  datasep
*
*   DESCRIPTION:  This algorithm was developed to separate the raw BAT tool
*     data into three files.  The first file is composed of the revolution
*     data including the revolution data headers.  The second file is a
*     collection of pointers, depths, and record lengths for the data file.
*     A final file containing all the log header records detected is
*     produced.
*
*   NOTES:  The last record in the input file will not be processed as it
*     does not have a terminating keystring.
*
*     When the data_point structure is full, that information is written
*     to the dp file.  The counters are reset (see data_point_index below)
*     and the structure refilled.  The log_head structure is not handled
*     in the same way.  It is assumed that it will never be completely
*     filled.  NO_LOG_HEAD should therefore be made sufficiently large to
*     prevent any possibility of overflow.
*
*     Description of the various indices & pointers used in this algorithm:
*
*           Name            Relative to          Description
*        cur_rec_index     input_buf          Identifies first byte of the
*                                             current record in the input_buf.
*        next_rec_index    input_buf          Identifies first byte of the
*                                             next record in the input_buf.
*        data_point_index  data_point struct  Specifies which structure elem
*                                             to store current rev data info.
*                                             This count is subject to roll-
*                                             over.
*        log_head_ptr      input file         Used to specify where in the
*                                             input file the log header
*                                             records were found.
*        data_rec_ptr      data output file   Used to determine where in the
*                                             data output file a data
*                                             record is located.
*
*   DEVELOPED BY: Troy K. Moore
*   DATE:         9/29/87
*
*   REVISION NOTES:
*     11/6/87  TKM   Changed the calling convention of get_depth.
*
*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*
*/

#include    <stdio.h>
#include    <fcntl.h>
#include    <sys\types.h>
#include    <sys\stat.h>
#include    <io.h>
```

Appendix F - DATASEP Source Code

```
#include    <string.h>

#define     VER                 "1.11"      /* software version no. */
#define     VER_DATE            "11/9/87"   /* version date */

#define     IN_BUF_LEN          32000       /* input buffer length ( < 32K) */
#define     INIT_READ_LEN       2000        /* bytes read for find_first_rec */
#define     NO_DATA_PTRS        1024        /* length of array of data ptrs */
#define     NO_LOG_HEAD         64          /* length of array of log h ptrs */

#define     LEN                 80          /* file names buffers length */
#define     CR                  0x0d        /* carriage return */

#define     DATA_FILE_EXT       "RAW"       /* default data file ext */
#define     DATA_POINTER_EXT    "DPF"       /* default data ptrs file ext */
#define     LOG_HEAD_EXT        "LHF"       /* default log header rec file ext */

#define     DEPTH_BYTES         8           /* no bytes in depth string */
#define     DEPTH_OFFSET        8           /* offset from start of rev header */
#define     DEPTH_DIGITS        2           /* no places to right of decimal point */
                                            /* on depth counter, 3 for WBK metric data */

struct rec_info   {
                     long int    address;
                     float       depth;
                     int         length;
                  } ;

struct head_info  {
                     long int    address;
                     int         length;
                  } ;

main()

{

   static char    input_buf[IN_BUF_LEN];

   int         fhand_in, fhand_data, fhand_datp, fhand_logh;
   int         cur_rec_type, next_rec_type, cur_rec_index, next_rec_index;
   int         err_flag, eof_flag, log_head_count;
   int         rec_length, no_bytes_read, data_point_index;

   long int    log_head_ptr, data_rec_ptr;
   long int    err_flagl, data_rec_count;

   static struct rec_info  data_point[NO_DATA_PTRS];
   static struct head_info log_head[NO_LOG_HEAD];
```

Appendix F - DATASEP Source Code

```
   void        name_gen();
   int         file_prep(), find_first_rec(), find_next_rec();
   int         save_log_heads();
   long int    lseek();
   float       get_depth();


/* Print out program header information */

   printf("\n\n");
   printf("************************************************************\n");
   printf("*          BAT Tool Data Separation Algorithm              *\n");
   printf("*         ESS-4, Los Alamos National Laboratory            *\n");
   printf("************************************************************\n");
   printf("*                Developed by Troy K. Moore                *\n");
   printf("*              Version: %4s  Dated: %8s           *\n",VER,VER_DATE);
   printf("************************************************************\n");

/* Obtain the name of the input file from the user, then open the input
   file and create the output files */

   err_flag = file_prep(&fhand_in,&fhand_data,&fhand_datp,&fhand_logh);
   if (err_flag == -1)
   {
      exit(1);
   }

/* Find the first record in the input file */

   err_flag = find_first_rec(fhand_in,input_buf,&next_rec_index,&next_rec_type);
   if (err_flag == -1)
   {
      exit(2);
   }

/* Adjust the file pointer to the start of the first record */

   err_flagl = lseek(fhand_in,(long int)next_rec_index,SEEK_SET);
   if (err_flagl == -1L)
   {
      perror("\ndatasep: error setting file pointer at startup");
      exit(3);
   }

/* Initialize counters and pointers */

   log_head_count = 0;
   data_rec_count = 0L;

   log_head_ptr =  (long int) next_rec_index;
   data_rec_ptr = 0L;
```

```
/* >>>>>>>>>>>> Begin main processing loop <<<<<<<<<<<<<<<< */

   eof_flag = 0;

   while (eof_flag == 0)
   {

/* Fill the input buffer, if not as many bytes were read as were requested,
   set the end of file flag */

      no_bytes_read = read(fhand_in,input_buf,IN_BUF_LEN);
      if (no_bytes_read != IN_BUF_LEN)
         eof_flag = 1;

/* Re-set error flag for entry into following loop */

      err_flag = 0;

/* Begin the loop for processing the current contents of the input buffer */

      for (next_rec_index = 0; next_rec_index < IN_BUF_LEN && err_flag != 1; )
      {

/* Make the next record the current record */

         cur_rec_index = next_rec_index;
         cur_rec_type = next_rec_type;

/* Find the next record in the input buffer */

         err_flag = find_next_rec(input_buf,cur_rec_index,no_bytes_read,
                        &next_rec_type,&next_rec_index);

         if (err_flag != 1)      /* next rec is within buffer */
         {

/* Calculate the current record's length */

            rec_length = next_rec_index - cur_rec_index;

/* Save the log header record location and length information */

            if (cur_rec_type == 0)
            {
               log_head[log_head_count].address = log_head_ptr;
               log_head[log_head_count].length = rec_length;
               log_head_count++;
            }
```

```
/* Save the revolution data record location and length information */

            else
            {
               data_point_index = data_rec_count % NO_DATA_PTRS;
               data_point[data_point_index].address = data_rec_ptr;
               data_point[data_point_index].depth =
                     get_depth(&input_buf[cur_rec_index+DEPTH_OFFSET]);
               data_point[data_point_index].length = rec_length;
               data_rec_count++;
               data_rec_ptr += (long int) rec_length;

/* If the revolution data pointer buffer is full, write it to the
   pointer file */

               if ((data_rec_count != 0) &&
                     ((data_rec_count % NO_DATA_PTRS) == 0))
               {
                  err_flag = write(fhand_datp,(char *)data_point,sizeof(data_point));
                  if (err_flag == -1)
                  {
                     perror("\ndatasep: error writing data pointers");
                     exit(5);
                  }
               }

               err_flag = write(fhand_data,&input_buf[cur_rec_index],rec_length);
               if (err_flag == -1)
               {
                  perror("\ndatasep: error writing to data output file");
                  exit(6);
               }

            }

/* Update the pointer to the log header records */

            log_head_ptr += (long int) rec_length;
         }

         else                    /* next rec not in buffer */
         {
```

```
/* If end of buffer is detected (by find_next_rec) but it is not the end of
   the file, re-position the file pointer for the next read */

            if (eof_flag == 0)
            {
               err_flagl = lseek(fhand_in,(long int)(cur_rec_index-no_bytes_read),SEEK_CUR);
               if (err_flagl == -1)
               {
                  perror("\ndatasep: error re-setting file pointer");
                  exit(4);
               }
            }

         }

      }

   }

/* >>>>>>>>>> End of main processing loop <<<<<<<<<<<<<<<< */

/* Write the remaining revolution data pointers to the pointer file */

   if ((data_rec_count % NO_DATA_PTRS) != 0)
   {
      err_flag = write(fhand_datp,(char *)data_point,
         (unsigned int)(sizeof(struct rec_info)*(data_rec_count%NO_DATA_PTRS)));
      if (err_flag == -1)
      {
         perror("\ndatasep: error writing data pointers");
         exit(7);
      }
   }

/* Write all the log header information to the log header file */

   err_flag = save_log_heads(fhand_in,fhand_logh,input_buf,log_head,
                             log_head_count);
   if (err_flag != 0)
   {
      exit(8);
   }

/* Print out a processing summary */
   printf("\n\n");
   printf("*****************************************************************\n");
   printf("*                 Data Separation Complete                      *\n");
   printf("*****************************************************************\n");
   printf("*     Revolution data records processed:  %9ld        *\n",
          data_rec_count);
   printf("*          Log header records processed:      %4d        *\n",
          log_head_count);
   printf("*****************************************************************\n");
```

```
/* Close all files that were open */

   close(fhand_in);
   close(fhand_data);
   close(fhand_datp);
   close(fhand_logh);

}
/*
*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*
*
*  MODULE NAME:  file_prep
*
*  DESCRIPTION:  This function is designed to obtain the name of the
*    input file from the users and open the file for use.  Based on the
*    input file name, default file names are created for the data, data
*    pointer, and log header files.  The user is queried regarding these
*    default names and has the chance to specify different ones.  The
*    output files are then created for use by the calling procedure.
*
*  NOTES:  Passed parameters
*          none
*          Returned parameters
*          ifh  - file handle to input file (pointer)
*          dfh  - file handle to data file (pointer)
*          dpfh - file handle to data pointer file (pointer)
*          lhfh - file handle to log header file (pointer)
*          Error flag
*             0 - successful completion
*            -1 - error encountered during file manipulation
*
*    Currently, this function will not support the redirection of input
*    for the calling routine.  See function getch.
*
*  DEVELOPED BY: Troy K. Moore
*  DATE:         7/28/87
*
*  REVISION NOTES:
*
*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*
*/


int file_prep(ifh,dfh,dpfh,lhfh)

int   *ifh, *dfh, *dpfh, *lhfh;

{

   char     input_file[LEN], data_file[LEN], point_file[LEN], log_file[LEN];
   int      answer, flag, count;
```

Appendix F - DATASEP Source Code

```
/* Prepare the input file */

   flag = -1;

   while (flag == -1)
   {
      printf("\nEnter input file name:  ");
      scanf("%s",input_file);
      printf("\n");

      flag = open(input_file, O_RDONLY|O_BINARY);
      if (flag == -1)
      {
         perror("\nfile_prep: error opening input file");
      }

   }

/* Initialize file handle for input file */

   *ifh = flag;

/* Generate the default output filenames based on input filename */

   name_gen(input_file,data_file,point_file,log_file);

/* Check for user approval of the default names */

   printf("\nDefault output file names are as follows:\n");

   printf("\tData file:  ");
   for (count = 0; flag = putchar(data_file[count]) != NULL; count++);

   printf("\n\tData pointer file:  ");
   for (count = 0; flag = putchar(point_file[count]) != NULL; count++);

   printf("\n\tLog header file:  ");
   for (count = 0; flag = putchar(log_file[count]) != NULL; count++);

   printf("\n\nTo accept these file names, hit the Enter key.\n");
   printf("To generate alternate file names, hit any other key.\n");

   answer = getch();
```

```
   if (answer != CR)
   {
      printf("\nEnter data file name:  ");
      scanf("%s",data_file);

      printf("Enter data pointer file name:  ");
      scanf("%s",point_file);

      printf("Enter log header file name:  ");
      scanf("%s",log_file);
   }

/* Create the output files */

   flag = open(data_file,O_CREAT|O_EXCL|O_WRONLY|O_BINARY,S_IWRITE|S_IREAD);
   if (flag == -1)
   {
      perror("\nfile_prep: error opening data file");
      return(-1);
   }

   *dfh = flag;

   flag = open(point_file,O_CREAT|O_EXCL|O_WRONLY|O_BINARY,S_IWRITE|S_IREAD);
   if (flag == -1)
   {
      perror("\nfile_prep: error opening data pointer file");
      return(-1);
   }

   *dpfh = flag;

   flag = open(log_file,O_CREAT|O_EXCL|O_WRONLY|O_BINARY,S_IWRITE|S_IREAD);
   if (flag == -1)
   {
      perror("\nfile_prep: error opening log header file");
      return(-1);
   }

   *lhfh = flag;

/* Return to the calling procedure */

   return(0);

}
```

Appendix F - DATASEP Source Code

```
/*
*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*
*
*  MODULE NAME:  namegen
*
*  DESCRIPTION:  This function is designed to generate the default
*    filenames for the data, data pointer, and log header files.
*
*  NOTES:  Passed parameters
*          input - input file name string (pointer)
*          Returned parameters
*          data_f  - default filename for data file (pointer)
*          data_pf - default filename for data pointer file (pointer)
*          log_hf  - default filename for log header file (pointer)
*

*  DEVELOPED BY: Troy K. Moore
*  DATE:         7/28/87
*
*  REVISION NOTES:
*
*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*
*/


void name_gen(input,data_f,data_pf,log_hf)

char     input[], data_f[], data_pf[], log_hf[];

{

   char     temp1[80], *period_ptr;
   int      length;


/* Copy input string into temp buffer */

   strcpy(temp1,input);

/* Convert input string to upper case */

   strupr(temp1);

/* Find the final '.' in the input string */

   period_ptr = strrchr(temp1,'.');
```

```
/* Pre-process any strings that did not contain a '. ' */

   if (period_ptr == NULL)
   {
      length = strlen(templ);
      templ[length+1] = NULL;
      templ[length] = '.';
   }

/* Truncate existing extension */

   else
   {
      *(period_ptr + 1) = NULL;
   }

/* Copy the filenames */

   strcpy(data_f,templ);
   strcpy(data_pf,templ);
   strcpy(log_hf,templ);

/* Append the file extensions */

   strcat(data_f,DATA_FILE_EXT);
   strcat(data_pf,DATA_POINTER_EXT);
   strcat(log_hf,LOG_HEAD_EXT);

   return;

}


/*
*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*
*
*  MODULE NAME:  find_first_rec
*
*  DESCRIPTION:  This module is designed to determine where the first
*     record is located in reference to the start of the input file.  An
*     offset, record type, and error flag are returned.
*
*  NOTES:  Passed parameters
*          f_hand - file handle associated with the input file (int)
*          buf    - buffer to read into (pointer)
*          Returned parameters
*          location - an offset into buf where the first char of the
*                     initial record is located (pointer)
*          type_rec - flag indicating what type of record found (pointer)
*                          0 - log header
*                          1 - rev data
```

```
*           Error flag (int)
*                 0 - successful execution
*                 1 - error detected while locating first record
*                -1 - read error
*
*   DEVELOPED BY: Troy K. Moore
*   DATE:         7/24/87
*
*   REVISION NOTES:
*     7/29/87  Changed location, start, and rec_loc from long int to int.
*
*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*
*/

int find_first_rec(f_hand,buf,location,type_rec)

char        buf[];
int         f_hand, *location, *type_rec;

{

   int         read_bytes, err_fnr, rec_type;
   int         start, rec_loc;

   int         find_next_rec();


/* Read a chunk of data into buf starting at the very first of the input file */

   read_bytes = read(f_hand,buf,INIT_READ_LEN);
   if (read_bytes == -1)
   {
      perror("\nfind_first_rec: error failed in initial read");
      return(-1);
   }

/* Find the first record */

   start = -1;                  /* for case first char is start of keystring */

   err_fnr = find_next_rec(buf,start,read_bytes,&rec_type,&rec_loc);
   if (err_fnr == 1)
   {
      return(1);
   }

/* Return information obtained regarding first record */

   *location = rec_loc;
   *type_rec = rec_type;
   return(0);

}
```

Appendix F - DATASEP Source Code

```
/*
*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*
*
*   MODULE NAME:  find_next_rec
*
*   DESCRIPTION:  This module is designed to, given the starting location
*     of the current record, search the input buffer to find and determine
*     the type of the next record.  An error is returned if the end of
*     the input buffer is reached before the next record is identified.
*
*   NOTES:  Passed parameters
*           in_buf - name of buffer in which to find next record
*                    (pointer)
*           offset - in_buf element index that specified the first byte
*                    of the current record (int)
*           no_byt - no. of bytes read into in_buf during last read, used
*                    to determine where end of buffer is relative
*                    to bytes being checked (int)
*           Returned parameters
*           type     - Flag to specify what type of record found (pointer)
*                             0 - log header
*                             1 - rev. data
*           next_rec - first byte of the next record (pointer)
*           Error flag (int)
*                       0 - no error detected
*                       1 - end of buffer detected
*
*   DEVELOPED BY: Troy K. Moore
*   DATE:         9/23/87
*
*   REVISION NOTES:
*
*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*
*/


int find_next_rec(in_buf,offset,no_byt,type,next_rec)

char        in_buf[];
int         *type, no_byt, offset, *next_rec;

{
   static char    root[] = "Sabis";
   static char    post_root_1[] = " Field Tape";
   static char    post_root_2[] = "log";

   int      count, xcount;
```

```
/* Check for match as long as do not over-run end of buffer */

   for (count=offset+1; count < (no_byt/sizeof(char)); )
   {
      if (in_buf[count] == root[0])       /* match first char */
      {

/* Check each char against root string */

         for (xcount=0; xcount < strlen(root) && in_buf[count] == root[xcount] &&
            count < (no_byt/sizeof(char)); xcount++, count++)
            ;

/* Check first char of post_root_1 string if all chars in root string matched */

         if (in_buf[count] == post_root_1[0] && xcount == strlen(root))
         {

/* Check each char against post_root_1 string */

            for (xcount=0; xcount < strlen(post_root_1) &&
               in_buf[count] == post_root_1[xcount] &&
               count < (no_byt/sizeof(char)); xcount++, count++)
               ;

/* If all matched then a log header has been found */

            if (xcount == strlen(post_root_1))
            {
               *next_rec = count - (strlen(root) + strlen(post_root_1));
               *type = 0;
               return(0);
            }
         }

/* Check first char of post_root_2 string if all chars in root string matched */

         else if (in_buf[count] == post_root_2[0] && xcount == strlen(root))
         {

/* Check each char against post_root_2 string */

            for (xcount=0; xcount < strlen(post_root_2) &&
               in_buf[count] == post_root_2[xcount] &&
               count < (no_byt/sizeof(char)); xcount++, count++)
```

```
/* If all matched then a revolution header has been found */

            if (xcount == strlen(post_root_2))
            {
               *next_rec = count - (strlen(root) + strlen(post_root_2));
               *type = 1;
               return(0);
            }
         }
      }

/* Character did not match first char in root */

      else
         count++;
   }

/* Reached only when the end of buffer is encountered */

   return(1);

}


/*
*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*
*
*   MODULE NAME:  get_depth
*
*   DESCRIPTION:  This routine is designed to convert the depth string
*     found in the revolution header to a floating point value.
*
*   NOTES:  Passed parameters
*           dep_start - start byte of depth string (pointer)
*           Returned parameters
*           converted value
*
*     Note that the depth is only valid to the number of places read from
*     the depth unit.
*
*   DEVELOPED BY: Troy K. Moore
*   DATE:         9/25/87
*
*   REVISION NOTES:
*     11/9/87  TKM   Bug found in handling the decimal point, calling
*                    sequence simplified by using pointers.  Changed the
*                    conversion method to include division.
*
*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*
*/

float get_depth(dep_start)

char     *dep_start;
```

```
{
   char        dbuf[DEPTH_BYTES+1];

   int         count;

   long        temp;

   float       value;

   long        atol();
   double      pow();


/* Terminate string buffer */

   dbuf[DEPTH_BYTES] = '\0';

/* Move depth string into a null terminated string */

   for (count=0; count < DEPTH_BYTES; count++,dep_start++)
      dbuf[count] = *dep_start;

/* Convert to long value and insert decimal point */

   temp = atol(dbuf);
   value = (float)((double)temp/pow((double)10.0,(double)DEPTH_DIGITS));

   return(value);

}

/*
*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*
*
*   MODULE NAME:  save_log_heads
*
*   DESCRIPTION:  This function is designed to write all the log headers
*     found during execution of DATASEP to the log header file.  The
*     actual address and record length of the log header relative to the
*     input file are added to the start of each log header.
*
*   NOTES:  Passed parameters
*           in_fh       - input file handle (int)
*           lh_fh       - log header file handle (int)
*           in_buf      - buffer used to hold log header records (pointer)
*           info_struct - pointer to the array of structure elements
*                         contains the log header address and length
*                         information (pointer)
*           lh_count    - number of file headers detected (int)
*           Returned parameters
*           none
```

```
*           Error flag (int)
*              0 - successful execution
*             -1 - file manipulation error
*
*   DEVELOPED BY: Troy K. Moore
*   DATE:         7/30/87
*
*   REVISION NOTES:
*     9/23/87  Modified calculation of struct_len (TKM).
*
*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*
*/

int save_log_heads(in_fh,lh_fh,in_buf,info_struct,lh_count)

char              in_buf[];
int               in_fh, lh_fh, lh_count;
struct head_info  *info_struct;

{
   int         count, flag, struct_len;
   long int    flagl;

   long int    lseek();


   struct_len = sizeof(struct head_info);

/* Process all log headers that have been detected */

   for (count = 0; count < lh_count; count++, info_struct++)
   {

/* Position the file pointer to the first byte of the log header */

      flagl = lseek(in_fh,info_struct->address,SEEK_SET);
      if (flagl == -1L)
      {
         perror("\nsave_log_heads: error setting file pointer");
         return(-1);
      }

/* Read the log header into the input buffer */

      flag = read(in_fh,in_buf,info_struct->length);
      if (flag != info_struct->length)
      {
         perror("\nsave_log_heads: error reading log header from input file");
         return(-1);
      }
```

```
/* Write the header location (relative to the input file) and length
   to the log header file */

      flag = write(lh_fh,info_struct,struct_len);
      if (flag != struct_len)
      {
         perror("\nsave_log_heads: error writing log header specifics");
         return(-1);
      }

/* Write the actual log header information to the log header file */

      flag = write(lh_fh,in_buf,info_struct->length);
      if (flag != info_struct->length)
      {
         perror("\nsave_log_heads: error writing log header data");
         return(-1);
      }

   }

   return(0);

}
```

Appendix G - DPDUMP Source Code

```
/*
*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*
*
*   MODULE NAME:  dpdump
*
*   DESCRIPTION:  This is a utility program that provides the capability
*     to dump the contents of a data pointer file produced by DATASEP.
*     The output of this algorithm, by default, is displayed on the console.
*     Using redirection, a hardcopy can be generated on the printer.
*
*     Any records that have lengths > midpoint_flag + flag_variance or
*     < midpoint_flag + flag_variance (both parameters are user specified)
*     are starred in the output.
*
*   NOTES:
*     The implementation of this utility is such that it reads and processes
*     just enough data at one time to produce one page of output.
*
*     Code is designed to accept only command line arguments.  The command
*     line syntax is as follows:
*
*           dpdump input_file midpoint_flag flag_variance
*
*   DEVELOPED BY: Troy K. Moore
*   DATE:         9/29/87
*
*   REVISION NOTES:
*
*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*
*/

#include    <stdio.h>

#define     HEAD_LINES     8          /* no header lines in output */
#define     FOOT_LINES     2          /* bottom of page margin */
#define     TOTAL_LINES    66         /* no lines on a page */

#define     BEEP           '\007'

main(argc,argv)

int      argc;
char     *argv[];

{

   char        input_fname[60];

   int         print_lines, flag_point, flag_var, flag_low, flag_hi;
   int         bytes_read, col1_end, col2_end, count1, count2, recs_read;

   long int    last_rec, first_rec;

   struct      rec_form    {
```

```
                    long int    offset;
                    float       depth;
                    int         length;
                 };

   static struct rec_form      rec_buf[(TOTAL_LINES-(HEAD_LINES+FOOT_LINES))*2];

   FILE         *in_stream;



/* Prompt for input if command line arguments not used */

   if (argc !=4)
   {
      fprintf(stderr,"\nCommand line syntax:\n\n");
      fprintf(stderr,"    dpdump input_file midpoint variance%c%c\n",BEEP,BEEP);
      exit(-1);
   }

/* Convert the command line arguments */

   else
   {
      flag_point = abs(atoi(argv[2]));
      flag_var = abs(atoi(argv[3]));
   }

/* Open the input file */

   in_stream = fopen(argv[1],"rb");
   if (in_stream == NULL)
   {
      perror("\ndpdump: error opening input file");
      exit(-1);
   }

/* Prepare for entering the main control loop */

   flag_low = flag_point - flag_var;
   flag_hi = flag_point + flag_var;
   print_lines = TOTAL_LINES - (HEAD_LINES + FOOT_LINES);
   last_rec = 0L;

/* Pre-load the buffer */

   bytes_read = fread((char *)rec_buf,sizeof(char),sizeof(rec_buf),in_stream);
```

```
/* Begin main control loop */

   while (bytes_read > 0)
   {

/* Print out the header */

      printf("\nData Pointer File Dump Utility, Version 1.0\n\n");
      printf("  File name: %-40.40s  Flag: %d +/-%d\n\n",
         argv[1],flag_point,flag_var);
      printf("Record  Offset            Record | Record  Offset            Record\n");
      printf("Number  in hex    Depth   Length | Number  in hex    Depth   Length\n");
      printf("-----------------------------------+-----------------------------------\n");

/*  Determine limits based on previous processing and no bytes read */

      recs_read = bytes_read/sizeof(struct rec_form);
      first_rec = last_rec + 1L;
      last_rec = first_rec + recs_read - 1L;

      col1_end = (recs_read > print_lines) ? print_lines : recs_read;
      col2_end = (recs_read - col1_end > print_lines) ?
                     print_lines : recs_read - col1_end;

/* Print out data on a line by line basis */

      for (count1 = 0, count2 = col1_end; count1 < col1_end; count1++, count2++)
      {
         if (rec_buf[count1].length > flag_hi || rec_buf[count1].length < flag_low)
            printf("%6ld  %8lx  %8.2f   %4d* | ",first_rec+count1,
               rec_buf[count1].offset,rec_buf[count1].depth,rec_buf[count1].length);

         else
            printf("%6ld  %8lx  %8.2f   %4d  | ",first_rec+count1,
               rec_buf[count1].offset,rec_buf[count1].depth,rec_buf[count1].length);

         if (count2 < col2_end+col1_end)
         {
            if (rec_buf[count2].length > flag_hi || rec_buf[count2].length < flag_low)
               printf("%6ld  %8lx  %8.2f   %4d*",first_rec+count2,
                  rec_buf[count2].offset,rec_buf[count2].depth,rec_buf[count2].length);

            else
               printf("%6ld  %8lx  %8.2f   %4d",first_rec+count2,
                  rec_buf[count2].offset,rec_buf[count2].depth,rec_buf[count2].length);
         }

         printf("\n");        /* terminate line with LF */

      }

      printf("\f");           /* terminate page with FF */
```

```
/* Fill the buffer for the next page */

   bytes_read = fread((char *)rec_buf,sizeof(char),sizeof(rec_buf),in_stream);

   }

/* Processing complete, close the input file and exit */

   fclose(in_stream);

   exit(0);

}
```

Appendix H - BDINVRT Source Code

```
/*
*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*
*
*  MODULE NAME:  bdinvrt.c
*
*  DESCRIPTION:  This algorithm is designed to invert the data in a DP/
*    data file set to simulate the data being collected in a different
*    direction.  The data at the end of the file are simply moved to the
*    start of the file and vice versa.
*
*  NOTES:  This is a modified version of bdextrct.c.
*
*  DEVELOPED BY: Troy K. Moore
*  DATE:         11/19/87
*
*  REVISION NOTES:
*
*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*
*/

#include     <fcntl.h>
#include     <sys\types.h>
#include     <sys\stat.h>
#include     <stdio.h>

#define      BUF_LEN        2048
#define      FN_LEN         80
#define      VER            1.00

main()

{

   static char    dp_in_fn[FN_LEN], data_in_fn[FN_LEN];
   static char    dp_out_fn[FN_LEN], data_out_fn[FN_LEN];
   static char    io_buf[BUF_LEN];

   int            dp_in, data_in, dp_out, data_out;
   int            read_bytes, write_bytes, buf_size;

   long int       new_offset, seek_flag, rec_count;

   struct dp_recs     {
                        long int    offset;
                        float       depth;
                        int         length;
                      } dp_buf;

   long           lseek();
```

Appendix H - BDINVRT Source Code

```
/* Print out title information */

   printf("\nBAT Data Inversion Utility\n");
   printf("Developed by Troy K. Moore\n");
   printf("    Version %.2f\n",VER);

/* Obtain input filenames */

   printf("\n<Input Filenames>\n");
   printf("    DP File:    ");
   scanf("%s",dp_in_fn);

   printf("    Data File:  ");
   scanf("%s",data_in_fn);

/* Open the input files */

   dp_in = open(dp_in_fn,O_RDONLY|O_BINARY);
   if (dp_in == -1)
   {
      perror("\nError opening DP file");
      exit(-1);
   }

   data_in = open(data_in_fn,O_RDONLY|O_BINARY);
   if (data_in == -1)
   {
      perror("\nError opening data file");
      exit(-1);
   }

/* Obtain output filenames */

   printf("\n<Output Filenames>\n");
   printf("    DP File:    ");
   scanf("%s",dp_out_fn);

   printf("    Data File:  ");
   scanf("%s",data_out_fn);

/* Create the output files */

   dp_out = open(dp_out_fn,O_CREAT|O_BINARY|O_RDWR,S_IREAD|S_IWRITE);
   if (dp_out == -1)
   {
      perror("\nError creating DP output file");
      exit(-1);
   }
```

```
   data_out = open(data_out_fn,O_CREAT|O_BINARY|O_RDWR,S_IREAD|S_IWRITE);
   if (data_out == -1)
   {
      perror("\nError creating data output file");
      exit(-1);
   }

/* Prepare for entering main processing loop */

   new_offset = 0L;                          /* offset counter */
   buf_size = sizeof(dp_buf);                /* size of dp buffer */

   seek_flag = lseek(dp_in,(long int)(-buf_size),SEEK_END);
   if (seek_flag == -1L)
   {
      perror("\nError positioning DP file pointer");
      exit(-1);
   }

/* Enter main processing loop, exit when DP file pointer moves past start
   of the file */

   for (rec_count = 0; seek_flag >= 0L; rec_count++)
   {

/* Read the DP record to determine where the data record is located */

      read_bytes = read(dp_in,&dp_buf,buf_size);
      if (read_bytes != buf_size)
      {
         printf("\nError reading from DP input file\n");
         printf("%d bytes read, %d bytes requested\n",read_bytes,buf_size);
         exit(-1);
      }

/* Make sure all the data record will fit in the buffer */

      if (dp_buf.length > BUF_LEN)
      {
         printf("\nInput record length overflow error\n");
         printf("%d bytes in rec, %d bytes in buffer\n",dp_buf.length,BUF_LEN);
         exit(-1);
      }

/* Position data file pointer prior to reading data record */

      seek_flag = lseek(data_in,dp_buf.offset,SEEK_SET);
      if (seek_flag == -1L)
      {
         perror("\nError positioning data file pointer");
         exit(-1);
      }
```

```
/* Read the data record */

      read_bytes = read(data_in,io_buf,dp_buf.length);
      if (read_bytes != dp_buf.length)
      {
         printf("\nError reading from the data input file\n");
         printf("%d bytes read, %d bytes requested\n",read_bytes,dp_buf.length);
         exit(-1);
      }

/* Write the data record out to the output data file */

      write_bytes = write(data_out,io_buf,dp_buf.length);
      if (write_bytes != dp_buf.length)
      {
         printf("\nError writing to the data output file\n");
         printf("%d bytes written, %d bytes requested\n",write_bytes,dp_buf.length);
         exit(-1);
      }
      dp_buf.offset = new_offset;           /* modify offset in DP record */
      new_offset += dp_buf.length;          /* update offset */

/* Write DP record to the output DP file */

      write_bytes = write(dp_out,&dp_buf,buf_size);
      if (write_bytes != buf_size)
      {
         printf("\nError writing to the DP output file\n");
         printf("%d bytes written, %d bytes requested\n",write_bytes,buf_size);
         exit(-1);
      }

/* Reposition the DP file pointer to the record in front of the one
   just processed */

      seek_flag = lseek(dp_in,(long int)(-2*buf_size),SEEK_CUR);
   }

/* Output the processing statistics */

   printf("\n%ld records were processed\n",rec_count);

/* Close the files */

   close(dp_in);
   close(dp_out);

   close(data_in);
   close(data_out);

   exit(0);

}
```

Appendix I - BDEXTRCT Source Code

```
/*
*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*
*
*   MODULE NAME:  bdextrct.c
*
*   DESCRIPTION:  This algorithm is designed to extract a contiguous sub-
*     set of data from a BAT data set.  The offset values in the new DP
*     file must be adjusted.
*
*   NOTES:
*
*   DEVELOPED BY: Troy K. Moore
*   DATE:         11/19/87
*
*   REVISION NOTES:
*
*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*
*/

#include    <fcntl.h>
#include    <sys\types.h>
#include    <sys\stat.h>
#include    <stdio.h>

#define     BUF_LEN         2048
#define     FN_LEN          80
#define     VER             1.00

main()

{

   static char    dp_in_fn[FN_LEN], data_in_fn[FN_LEN];
   static char    dp_out_fn[FN_LEN], data_out_fn[FN_LEN];
   static char    io_buf[BUF_LEN];

   int            dp_in, data_in, dp_out, data_out;
   int            read_bytes, write_bytes, buf_size;

   long int       start_rec, end_rec, extract_recs, new_offset;
   long int       seek_flag, rec_count;

   struct dp_recs    {
                        long int    offset;
                        float       depth;
                        int         length;
                     } dp_buf;

   long           lseek();
```

Appendix I - BDEXTRCT Source Code

```
/* Print out title information */

   printf("\nBAT Data Extraction Utility\n");
   printf("Developed by Troy K. Moore\n");
   printf("    Version %.2f\n",VER);

/* Obtain input filenames */

   printf("\n<Input Filenames>\n");
   printf("    DP File:    ");
   scanf("%s",dp_in_fn);

   printf("    Data File:  ");
   scanf("%s",data_in_fn);

/* Open the input files */

   dp_in = open(dp_in_fn,O_RDONLY|O_BINARY);
   if (dp_in == -1)
   {
      perror("\nError opening DP file");
      exit(-1);
   }

   data_in = open(data_in_fn,O_RDONLY|O_BINARY);
   if (data_in == -1)
   {
      perror("\nError opening data file");
      exit(-1);
   }

/* Obtain output filenames */

   printf("\n<Output Filenames>\n");
   printf("    DP File:    ");
   scanf("%s",dp_out_fn);

   printf("    Data File:  ");
   scanf("%s",data_out_fn);

/* Create the output files */

   dp_out = open(dp_out_fn,O_CREAT|O_BINARY|O_RDWR,S_IREAD|S_IWRITE);
   if (dp_out == -1)
   {
      perror("\nError creating DP output file");
      exit(-1);
   }
```

```
   data_out = open(data_out_fn,O_CREAT|O_BINARY|O_RDWR,S_IREAD|S_IWRITE);
   if (data_out == -1)
   {
      perror("\nError creating data output file");
      exit(-1);
   }

/* Obtain first and last records to extract */

   printf("\n<Record Specifications>\n");
   printf("    Start extraction with record:  ");
   scanf("%ld",&start_rec);

   printf("    Final record to extract:       ");
   scanf("%ld",&end_rec);

/* Prepare for entering main processing loop */

   extract_recs = end_rec - start_rec + 1L;  /* no recs to extract */
   new_offset = 0L;                          /* offset counter */
   buf_size = sizeof(dp_buf);                /* size of dp buffer */

   seek_flag = lseek(dp_in,(long int)((start_rec-1L)*buf_size),SEEK_SET);
   if (seek_flag == -1L)
   {
      perror("\nError positioning DP file pointer");
      exit(-1);
   }

/* Enter main processing loop */

   for (rec_count = 0; rec_count < extract_recs; rec_count++)
   {

/* Read the DP record to determine where data record is located */

      read_bytes = read(dp_in,&dp_buf,buf_size);
      if (read_bytes != buf_size)
      {
         printf("\nError reading from DP input file\n");
         printf("%d bytes read, %d bytes requested\n",read_bytes,buf_size);
         exit(-1);
      }

/* Make sure all the data record will fit in the buffer */

      if (dp_buf.length > BUF_LEN)
      {
         printf("\nInput record length overflow error\n");
         printf("%d bytes in rec, %d bytes in buffer\n",dp_buf.length,BUF_LEN);
         exit(-1);
      }
```

```
/* Position data file pointer prior to reading data record */

      seek_flag = lseek(data_in,dp_buf.offset,SEEK_SET);
      if (seek_flag == -1L)
      {
         perror("\nError positioning data file pointer");
         exit(-1);
      }

/* Read the data record */

      read_bytes = read(data_in,io_buf,dp_buf.length);
      if (read_bytes != dp_buf.length)
      {
         printf("\nError reading from the data input file\n");
         printf("%d bytes read, %d bytes requested\n",read_bytes,dp_buf.length);
         exit(-1);
      }

/* Write the data record out to the output data file */

      write_bytes = write(data_out,io_buf,dp_buf.length);
      if (write_bytes != dp_buf.length)
      {
         printf("\nError writing to the data output file\n");
         printf("%d bytes written, %d bytes requested\n",write_bytes,dp_buf.length);
         exit(-1);
      }

      dp_buf.offset = new_offset;           /* modify offset in DP record */
      new_offset += dp_buf.length;          /* update offset */

/* Write the DP record to the output DP file */

      write_bytes = write(dp_out,&dp_buf,buf_size);
      if (write_bytes != buf_size)
      {
         printf("\nError writing to the DP output file\n");
         printf("%d bytes written, %d bytes requested\n",write_bytes,buf_size);
         exit(-1);
      }
   }

/* Output the processing statistics */

   printf("\n%ld records were extracted\n",rec_count);

/* Close the files */

   close(dp_in);
   close(dp_out);

   close(data_in);
   close(data_out);
```

```
    exit(0);

}
```

Appendix J - BATD Source Code

```
==========================================================================
                           File: BATDDEFS.H
==========================================================================

/* Version information */

#define  VER          "1.01"
#define  VER_DATE     "12/7/87"

/* Run modes */

#define  GO           0
#define  STOP         1

/* Data modes */

#define  BOTH         0
#define  AMPLITUDE    1
#define  T_TIME       2

/* Modes of display */

#define  FRAME_MODE   0
#define  SCAN_MODE    1

/* Frame reference switches */

#define  RECORD       0
#define  DEPTH        1

/* Display channel switches */

#define  CHAN_0       0
#define  CHAN_1       1

/* Default parameters */

#define  DEF_START    1
#define  DEF_LCOUNT   480
#define  DEF_ECOUNT   128
#define  DEF_INC      1

/* Record processing parameters */

#define  NO_RECS_PROC   1024

/* Revolution record structure */

#define  NO_DATA_VALS   128         /* Number of data points per record */
#define  REV_HEAD_LEN   48          /* Number of bytes in rev header */
```

```
/* Display parameters */

#define  NO_DISP_LINES  480
#define  NO_DISP_ELEMS  512

/* Depth record select variance criteria */

#define  CHECK_BAND      1.0         /* see depth_sel_rec */

/* Length of filename strings */

#define  FNAME_LEN       40
```

File: BATDGLOB.H

```
char      data_fname[FNAME_LEN], dp_fname[FNAME_LEN];

int       data_type, disp_mode, frame_ref;

int       sline_start, sline_count, sline_end, sline_inc;
int       selem_start, selem_count, selem_end, selem_inc;

int       dc0_line_start, dc0_elem_start, dc1_line_start, dc1_elem_start;

int       temp1, temp2, temp3;

int       disp_chan, run_flag, dfh, dpfh, amp_sf, tt_sf;

long int     first_rec, last_rec;

float        first_depth, last_depth;

struct rec_info    {
                   long int    rec_offset;
                   float       depth;
                   int         rec_length;
                 };
```

File: BATDWND.H

```
#include <wfd.h>
#include <wfd_glob.h>

DFORMPTR     fn_form, cnt_form, end_form, menu_start;
DFORMPTR     disp_form, sf_form;
```

```
WINDOWPTR   stat_wind, rinfo_wind, error_wind, title_wind;

WINDOW      status_wn, rinfo_wn, error_wn, title_wn;


===========================================================================
                         File: BATDXTRN.H
===========================================================================


extern char     data_fname[], dp_fname[];

extern int      data_type, disp_mode, frame_ref;

extern int      sline_start, sline_count, sline_end, sline_inc;
extern int      selem_start, selem_count, selem_end, selem_inc;

extern int      dc0_line_start, dc0_elem_start, dc1_line_start, dc1_elem_start;

extern int      temp1, temp2, temp3;

extern int      disp_chan, run_flag, dfh, dpfh, amp_sf, tt_sf;

extern long int       first_rec, last_rec;

extern float          first_depth, last_depth;

struct rec_info   {
                      long int    rec_offset;
                      float       depth;
                      int         rec_length;
                  };


===========================================================================
                          File: BATDXWND.H
===========================================================================


#include <wfd.h>

extern DFORMPTR       fn_form, cnt_form, end_form, menu_start;
extern DFORMPTR       disp_form, sf_form;

extern WINDOWPTR      stat_wind, rinfo_wind, error_wind, title_wind;

extern WINDOW         status_wn, rinfo_wn, error_wn, title_wn;
```

Appendix J - BATD Source Code

```
----------------------------------------------------------------------------
                              File: BATD.C
----------------------------------------------------------------------------


/*
*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*
*
*   MODULE NAME:  batd
*
*   DESCRIPTION:  This is the main module of the code designed for the
*     display of BAT tool data.  Extensive use of functions is used to
*     modularize implementation of this task.
*
*   NOTES:
*
*   DEVELOPED BY: Troy K. Moore
*   DATE:         10/7/87
*
*   REVISION NOTES:
*     11/12/87 TKM   Change name of DT-IRIS include file.
*     12/7/87  TKM   Add code to erase display channels.
*
*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*
*/

#define  WN_DEBUG

#include <batddefs.h>
#include <batdglob.h>
#include <batdwnd.h>

#include <\image\dt_iris\isdefs.c>

main()

{
   int          ret_flag, exit_val;
   static int          rec_length[NO_DISP_LINES];

   long int     fsel_rec, lsel_rec;
   static long int      rec_offset[NO_DISP_LINES];

   float        fsel_depth, lsel_depth;

   int          depth_sel_rec(), rec_sel_rec1(), rec_sel_rec2();
   void         init_parms(), window_setup(), fill_stat_wn(), menu_setup();
   void         recs_specs(), put_title();


   init_wfd();                /* Initialize windows for data */
   is_initialize();           /* Initialize the frame buffer */
   cls();                     /* Clear the display */
   init_parms();              /* Initialize all the parameters */
```

```
is_select_olut(0);           /* Select the output lut */
is_frame_clear(CHAN_0);      /* Erase display channel 0 */
is_frame_clear(CHAN_1);      /* Erase display channel 1 */
is_display(1);               /* Turn the display on */
is_select_output_frame(disp_chan); /* Select chan to display */
window_setup();              /* Define the windows and forms to be used */
put_title();                 /* Output the title block */
menu_setup();                /* Define the menu structure */
set_wn(stat_wind);           /* Place the status window on the display */
fill_stat_wn();              /* Write information to the status window */

while (run_flag == GO)
{
   ret_flag = mn_proc(0,menu_start);      /* Use the menu to obtain parameters */


   if (run_flag == GO)
   {
      if (frame_ref == RECORD)
      {
         if (sline_inc < 0)
            ret_flag = rec_sel_rec1(rec_offset,rec_length,&fsel_depth,
                  &lsel_depth);

         else
            ret_flag = rec_sel_rec2(rec_offset,rec_length,&fsel_depth,
                  &lsel_depth);
      }

      else
         ret_flag = depth_sel_rec(rec_offset,rec_length,&fsel_rec,&lsel_rec);

      if (ret_flag == 0)
         ret_flag = disp_recs(rec_offset,rec_length);

      if (ret_flag == 0)
         recs_specs(fsel_depth,lsel_depth,fsel_rec,lsel_rec);


   }


}

unset_wn(stat_wind);         /* Remove the status window from the display */
is_end();                    /* Terminating DT-IRIS call */

exit_val = (ret_flag != 0) ? -1 : 0;

exit(exit_val);

}
```

Appendix J - BATD Source Code

```
---------------------------------------------------------------------------
                          File: BATDSTUP.C
---------------------------------------------------------------------------


/*
*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*
*
*   MODULE NAME:  init_parms
*
*   DESCRIPTION:  This function is designed to initialize parameters at
*     the start of batd.
*
*   NOTES:  This function is called directly by batd.c.
*
*   DEVELOPED BY: Troy K. Moore
*   DATE:         10/5/87
*
*   REVISION NOTES:
*
*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*
*/

#define     WN_DEBUG

#include    <batdxtrn.h>
#include    <batdxwnd.h>
#include    <batddefs.h>

void init_parms()

{

/* General mode-type parameters */

   run_flag = GO;
   data_type = BOTH;
   disp_mode = FRAME_MODE;
   frame_ref = RECORD;

/* Source line and element parameters */

   sline_start = DEF_START;
   sline_count = DEF_LCOUNT;
   sline_inc = DEF_INC;
   sline_end = (sline_count - sline_start + 1)/sline_inc;

   selem_start = DEF_START;
   selem_count = DEF_ECOUNT;
   selem_inc = DEF_INC;
   selem_end = (selem_count - selem_start + 1)/selem_inc;
```

```
/* Display channel 0 line and element parameters */

   dc0_line_start = DEF_START;
   dc0_elem_start = DEF_START;

/* Display channel 1 line and element parameters */

   dc1_line_start = DEF_START;
   dc1_elem_start = DEF_START;

/* Initialize the file handles */

   dfh = 0;
   dpfh = 0;

/* Initialize the scale factors */

   amp_sf = 0;
   tt_sf = 0;

/* Initialize the first and last record numbers */

   first_rec = 1L;              /* this value is never changed */
   last_rec = 0L;

/* Initialize the first and last record depths */

   first_depth = 0.0;
   last_depth = 0.0;

   return;

}

/*
*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*
*
*   MODULE NAME:  menu_setup
*
*   DESCRIPTION:  This function defines the menu structure to be used in
*     obtaining parameters from the user.
*
*   NOTES:  This function is called directly by batd.c
*
*   DEVELOPED BY: Troy K. Moore
*   DATE:         10/5/87
*
*   REVISION NOTES:
*     12/7/87  TKM   Added code to not allow ESC to exit menu at top level.
*
*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*
*/
```

Appendix J - BATD Source Code

```
void  menu_setup()

{

    DFORMPTR     lev2_opt1, lev2_opt2;

    DFORMPTR     lev3_opt12, lev3_opt13, lev3_opt21;

    DFORMPTR     lev4_opt131;

    DFORMPTR     lev5_opt1311, lev5_opt1312, lev5_opt1313;

    DFIELDPTR    mnf_def();

    DFORMPTR     mnfm_def(), mnfm_dap();

    int          alev2_opt11(), alev2_opt22(), alev2_opt23();

    int          alev3_opt121(), alev3_opt122(), alev3_opt123(), alev3_opt132();
    int          alev3_opt211(), alev3_opt212();

    int          alev5_opt13111(), alev5_opt13112();
    int          alev5_opt13121(), alev5_opt13122();
    int          alev5_opt13131(), alev5_opt13132();

    int          arun(), aexit();


    char *p_lev1_opt1 =      "Specify source parameters";
    char *p_lev1_opt2 =      "Specify display parameters";
    char *p_lev1_opt3 =      "Display an image";
    char *p_lev1_opt4 =      "Exit to DOS";

    char *p_lev2_opt11 =     "Specify input filename";
    char *p_lev2_opt12 =     "Select type of data to display";
    char *p_lev2_opt13 =     "Select how data is treated";
    char *p_lev2_opt21 =     "Select channel to transfer data to and display";
    char *p_lev2_opt22 =     "Enter parameters for Channel 0";
    char *p_lev2_opt23 =     "Enter parameters for Channel 1";

    char *p_lev3_opt121 =    "Display amplitude data only";
    char *p_lev3_opt122 =    "Display travel time data only";
    char *p_lev3_opt123 =    "Display both types of data";

    char *p_lev3_opt131 =    "Display a single frame";
    char *p_lev3_opt132 =    "Scroll through all data";
    char *p_lev3_opt211 =    "Write to and display Channel 0";
    char *p_lev3_opt212 =    "Write to and display Channel 1";

    char *p_lev4_opt1311 =   "Specify first line to display";
    char *p_lev4_opt1312 =   "Specify first element to display";
    char *p_lev4_opt1313 =   "Reference to select line information";
```

```
    char *p_lev5_opt13111 = "Specify a # of lines to display";
    char *p_lev5_opt13112 = "Specify the last line to display";

    char *p_lev5_opt13121 = "Specify a # of elements to display";
    char *p_lev5_opt13122 = "Specify the last element to display";

    char *p_lev5_opt13131 = "Line specification referenced to depth";
    char *p_lev5_opt13132 = "Line specification referenced to record";

/* Set up the first level menu */

    menu_start = mnfm_def(0,0,1,80,LNORMAL,BDR_OP);

/* Do not allow user to ESC out top of menu */

    sfm_opt(MNTOPESCAPE,OFF,menu_start);

/*  All the level 2 menus will be pop up */

    lev2_opt1 = mnfm_dap(MNPOP,LNORMAL,BDR_LNP);
    lev2_opt2 = mnfm_dap(MNPOP,LNORMAL,BDR_LNP);

/* All menus level 3 and below are pull up */

    lev3_opt12 = mnfm_dap(MNPULL,LNORMAL,BDR_LNP);
    lev3_opt13 = mnfm_dap(MNPULL,LNORMAL,BDR_LNP);
    lev3_opt21 = mnfm_dap(MNPULL,LNORMAL,BDR_LNP);

    lev4_opt131 = mnfm_dap(MNPULL,LNORMAL,BDR_LNP);

    lev5_opt1311 = mnfm_dap(MNPULL,LNORMAL,BDR_LNP);
    lev5_opt1312 = mnfm_dap(MNPULL,LNORMAL,BDR_LNP);
    lev5_opt1313 = mnfm_dap(MNPULL,LNORMAL,BDR_LNP);

/* Level 1 menu */

    mnf_def(0,0,"Source",p_lev1_opt1,lev2_opt1,NULLFP,menu_start);
    mnf_def(0,23,"Destination",p_lev1_opt2,lev2_opt2,NULLFP,menu_start);
    mnf_def(0,50,"Run",p_lev1_opt3,NULLP,arun,menu_start);
    mnf_def(0,71,"Exit",p_lev1_opt4,NULLP,aexit,menu_start);

/* All menus under option 1, level 1 */

    mnf_def(0,0,"Filename    ",p_lev2_opt11,NULLP,alev2_opt11,lev2_opt1);
    mnf_def(1,0,"Data        ",p_lev2_opt12,lev3_opt12,NULLFP,lev2_opt1);
    mnf_def(2,0,"Mode        ",p_lev2_opt13,lev3_opt13,NULLFP,lev2_opt1);

    mnf_def(0,0,"Amplitude  ",p_lev3_opt121,NULLP,alev3_opt121,lev3_opt12);
    mnf_def(1,0,"Travel time",p_lev3_opt122,NULLP,alev3_opt122,lev3_opt12);
    mnf_def(2,0,"Both       ",p_lev3_opt123,NULLP,alev3_opt123,lev3_opt12);

    mnf_def(0,0,"Frame      ",p_lev3_opt131,lev4_opt131,NULLFP,lev3_opt13);
    mnf_def(1,0,"Scan       ",p_lev3_opt132,NULLP,alev3_opt132,lev3_opt13);
```

```
   mnf_def(0,0,"Line       ",p_lev4_opt1311,lev5_opt1311,NULLFP,lev4_opt131);
   mnf_def(1,0,"Element    ",p_lev4_opt1312,lev5_opt1312,NULLFP,lev4_opt131);
   mnf_def(2,0,"Reference  ",p_lev4_opt1313,lev5_opt1313,NULLFP,lev4_opt131);

   mnf_def(0,0,"Count      ",p_lev5_opt13111,NULLP,alev5_opt13111,lev5_opt1311);
   mnf_def(1,0,"End        ",p_lev5_opt13112,NULLP,alev5_opt13112,lev5_opt1311);

   mnf_def(0,0,"Count      ",p_lev5_opt13121,NULLP,alev5_opt13121,lev5_opt1312);
   mnf_def(1,0,"End        ",p_lev5_opt13122,NULLP,alev5_opt13122,lev5_opt1312);

   mnf_def(0,0,"Depth      ",p_lev5_opt13131,NULLP,alev5_opt13131,lev5_opt1313);
   mnf_def(1,0,"Record     ",p_lev5_opt13132,NULLP,alev5_opt13132,lev5_opt1313);

/* All menus under level 1, option 2 */

   mnf_def(0,0,"Select channel",p_lev2_opt21,lev3_opt21,NULLFP,lev2_opt2);
   mnf_def(1,0,"Parms - Chan 0",p_lev2_opt22,NULLP,alev2_opt22,lev2_opt2);
   mnf_def(2,0,"Parms - Chan 1",p_lev2_opt23,NULLP,alev2_opt23,lev2_opt2);

   mnf_def(0,0,"Channel 0  ",p_lev3_opt211,NULLP,alev3_opt211,lev3_opt21);
   mnf_def(1,0,"Channel 1  ",p_lev3_opt212,NULLP,alev3_opt212,lev3_opt21);

   return;

}


----------------------------------------------------------------------------
                          File: BATDWIND.C
----------------------------------------------------------------------------


/*
*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*
*
*  MODULE NAME:  window_setup
*
*  DESCRIPTION:  This function is designed to define the forms called up
*    via menu selections to obtain parameters from the user.  It also
*    sets up the status window and modifies the menu message window.
*
*  NOTES:  This function is called directly by batd.c
*
*  DEVELOPED BY: Troy K. Moore
*  DATE:         10/7/87
*
*  REVISION NOTES:
*    12/7/87  TKM   Added code to set form defaults at a global level.
*
*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*
*/
```

Appendix J - BATD Source Code

```
#define  WN_DEBUG

#include <batdxtrn.h>
#include <batdxwnd.h>
#include <batddefs.h>

void window_setup()


{
   DFORMPTR    fm_def();
   DFIELDPTR   fld_def();
   void        defs_wn(), mod_wn(), sfm_opt();


/* Set form option settings (see p 5-6, WFD manual) */

   se_fmopt(FMDATA,AUTOEXIT);

/* Configure a two line data form for obtaining file names */

   fn_form = fm_def(7,24,4,56,LNORMAL,BDR_DLNP);
   fld_def(0,0,"Filename:    ",FADJACENT,def_pic('X',FNAME_LEN-1),F_STRING,data_fname,fn_form);
   fld_def(1,0,"DP filename: ",FADJACENT,def_pic('X',FNAME_LEN-1),F_STRING,dp_fname,fn_form);

/* Configure three line data form for 'count' passing */

   cnt_form = fm_def(7,30,5,30,LNORMAL,BDR_DLNP);
   fld_def(0,0,"Start:      ",FADJACENT,"99999",F_INT,(char *)&temp1,cnt_form);
   fld_def(1,0,"Increment: ",FADJACENT,"999",F_INT,(char *)&temp2,cnt_form);
   fld_def(2,0,"Count:      ",FADJACENT,"99999",F_INT,(char *)&temp3,cnt_form);

/* Configure three line data form for 'end' passing */

   end_form = fm_def(7,30,5,30,LNORMAL,BDR_DLNP);
   fld_def(0,0,"Start:      ",FADJACENT,"99999",F_INT,(char *)&temp1,end_form);
   fld_def(1,0,"Increment: ",FADJACENT,"999",F_INT,(char *)&temp2,end_form);
   fld_def(2,0,"End:        ",FADJACENT,"99999",F_INT,(char *)&temp3,end_form);

/* Configure a two line data form for obtaining display location info */

   disp_form = fm_def(7,30,4,30,LNORMAL,BDR_DLNP);
   fld_def(0,0,"Starting line:    ",FADJACENT,"999",F_INT,(char *)&temp1,disp_form);
   fld_def(1,0,"Starting element: ",FADJACENT,"999",F_INT,(char *)&temp2,disp_form);

/* Configure a two line data form for obtaining scale factors */

   sf_form = fm_def(7,30,4,40,LNORMAL,BDR_DLNP);
   fld_def(0,0,"Amplitude scale factor:   ",FADJACENT,"99",F_INT,(char *)&temp1,sf_form);
   fld_def(1,0,"Travel time scale factor: ",FADJACENT,"99",F_INT,(char *)&temp2,sf_form);
```

```
/* Configure status window */

   stat_wind = &status_wn;
   defs_wn(stat_wind,15,0,10,80,BDR_LNP);
   sw_name(" Status Window ",stat_wind);
   sw_namelocation(BOTTOMCENTER,stat_wind);

/* Set up an error message window */

   error_wind = & error_wn;
   defs_wn(error_wind,5,5,4,70,BDR_LNP);
   sw_popup(ON,error_wind);
   sw_att(LRED,error_wind);
   sw_name(" Strike any key to return to menu ",error_wind);
   sw_namelocation(BOTTOMCENTER,error_wind);

/* Set up a message window for first/last record information */

   rinfo_wind = &rinfo_wn;
   defs_wn(rinfo_wind,4,15,4,50,BDR_LNP);
   sw_popup(ON,rinfo_wind);
   sw_att(LREVERSE,rinfo_wind);
   sw_name(" Strike any key to continue ",rinfo_wind);
   sw_namelocation(BOTTOMCENTER,rinfo_wind);

/* Set up a message window for title block */

   title_wind = &title_wn;
   defs_wn(title_wind,4,20,9,40,BDR_DLNP);
   sw_popup(ON,title_wind);
   sw_att(LNORMAL,title_wind);
   sw_name(" Strike any key to begin ",title_wind);
   sw_namelocation(BOTTOMCENTER,title_wind);

/* Modify the menu message window */

   se_mnmsg(MANUAL);
   sw_border(BDR_OP,&mnu_msgw);
   mod_wn(14,0,1,80,&mnu_msgw);

   return;

}


/*
*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*
*
*  MODULE NAME:  put_title
*
*  DESCRIPTION:  This function is designed to output a title block prior
*     to entering the menu section of the algorithm.
*
*  NOTES:  No parameters passed or returned.
```

```
*
*  DEVELOPED BY: Troy K. Moore
*  DATE:         10/7/87
*
*  REVISION NOTES:
*
*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*
*/


void put_title()

{
   int      temp;

   char *line1 = "   BAT Tool Data Display Utility\n";
   char *line2 = "    Developed by Troy K. Moore\n\n";
   char *line3 = "          Version: %4s\n";
   char *line4 = "          Dated: %8s";


/* Prepare for writing to the title window */

   set_wn(title_wind);
   mv_cs(1,0,title_wind);

/* Place information in the window */

   v_st(line1,title_wind);
   v_st(line2,title_wind);
   v_printf(title_wind,line3,VER);
   v_printf(title_wind,line4,VER_DATE);

/* Wait for input from the keyboard before removing window and beginning */

   temp = ki();
   unset_wn(title_wind);

   return;

}


/*
*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*
*
*  MODULE NAME:  fill_stat_wn
*
*  DESCRIPTION:  This function is designed to fill the status window
*    with initial parameter information at the start of code execution.
*
*  NOTES:  fill_stat_wn is called directly by batd.c
*
```

```
*   DEVELOPED BY: Troy K. Moore
*   DATE:         10/5/87
*
*   REVISION NOTES:
*
*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*
*/

void fill_stat_wn()

{
   int   count;

   void stat_wind_wrt();

   for (count = 1; count <= 8; count++)
      stat_wind_wrt(count);

   return;

}


/*
*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*
*
*   MODULE NAME:  stat_wind_wrt
*
*   DESCRIPTION:  stat_wind_wrt is designed to provide the ability to
*     randomly write a specific line in the status window using current
*     parameter values.
*
*   NOTES:  The status window is composed of 8 lines, number 1 thru 8.
*           The passed parameter, an int, may be from 1 to 8.
*
*   DEVELOPED BY: Troy K. Moore
*   DATE:         10/5/87
*
*   REVISION NOTES:
*     12/7/87  TKM   Added default case.
*
*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*
*/

void stat_wind_wrt(line_no)

int   line_no;

{
   void  mv_cs();
   char  *v_st();
   int   v_printf();
```

```
switch(line_no)
{
   case 1:                    /* Line 1 */
      mv_cs(0,0,stat_wind);
      v_st("Source",stat_wind);
      break;

   case 2:                    /* Line 2 */
      mv_cs(1,0,stat_wind);
      v_printf(stat_wind,
         " Data Fname.%-24.24s  DP Fname...%-24.24s",
         data_fname,dp_fname);
      break;

   case 3:                    /* Line 3 */
      mv_cs(2,0,stat_wind);
      v_printf(stat_wind,
         " Recs..First: %11d  Last: %61d        Depth.First: %7.1f Last: %7.1f",
         first_rec,last_rec,first_depth,last_depth);
      break;

   case 4:                    /* Line 4 */
      mv_cs(3,0,stat_wind);

      if (data_type == AMPLITUDE)
         v_printf(stat_wind," Data type..Amplitude                 Scale Factor....Amp: %2u   TT: %2u",
                  amp_sf,tt_sf);

      else if (data_type == T_TIME)
         v_printf(stat_wind," Data type..Travel time               Scale Factor....Amp: %2u   TT: %2u",
                  amp_sf,tt_sf);


      else
         v_printf(stat_wind," Data type..Both                      Scale Factor....Amp: %2u   TT: %2u",
                  amp_sf,tt_sf);


      break;

   case 5:                    /* Line 5 */
      mv_cs(4,0,stat_wind);

      if (disp_mode == FRAME_MODE && frame_ref == DEPTH)
         v_st(" Mode.......Frame  Ref...Depth       Destination",stat_wind);

      else if (disp_mode == FRAME_MODE && frame_ref == RECORD)
         v_st(" Mode.......Frame  Ref...Record      Destination",stat_wind);

      else
         v_st(" Mode.......Scan                     Destination",stat_wind);

      break;
```

```
      case 6:                     /* Line 6 */
         mv_cs(5,0,stat_wind);
         v_printf(stat_wind,
            "        Start Count Inc  End        Selected Channel......Channel %1d",
            disp_chan);

         break;

      case 7:                     /* Line 7 */
         mv_cs(6,0,stat_wind);
         v_printf(stat_wind,
            "   Line  %5d  %4d %3d %5d       Channel 0....Line: %3d   Elem: %3d",
            sline_start,sline_count,sline_inc,sline_end,
            dc0_line_start,dc0_elem_start);

         break;

      case 8:                     /* Line 8 */
         mv_cs(7,0,stat_wind);
         v_printf(stat_wind,
            "   Elem  %5d  %4d %3d %5d       Channel 1....Line: %3d   Elem: %3d",
            selem_start,selem_count,selem_inc,selem_end,
            dc1_line_start,dc1_elem_start);

         break;

      default:                    /* Passed parameter not supported */

         break;

      }

   return;

}


/*
*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*
*
*  MODULE NAME:  check_parms
*
*  DESCRIPTION:  check_parms was developed to check the parameters
*    selected by the user through the menu before being passed to the
*    record selection portion of the algorithm.  If this function does
*    detect an error, the error handler function is called to notify
*    the user.  He/she is then sent back to the menu to modify the
*    original parameters in order to eliminate the error condition.
*
*  NOTES:  Only the first error detected is processed.
*
```

```
*       Return value
*             0 - no errors found
*            -1 - error detected
*
*   DEVELOPED BY: Troy K. Moore
*   DATE:         10/7/87
*
*   REVISION NOTES:
*     12/7/87  TKM   Changed code to exit function any time an error is.
*                    identified.
*
*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*
*/

int   check_parms()

{
   void  error_handler();


/* Check the number of elements from the input */

   if (1 > selem_start || selem_start > NO_DATA_VALS)
   {
      error_handler(1);
      return(-1);
   }

/* The last element must be in the data set */

   if (selem_end < 1 || selem_end > NO_DATA_VALS)
   {
      error_handler(2);
      return(-1);
   }

/* The first element must be before the last element */

   if (selem_start > selem_end)
   {
      error_handler(3);
      return(-1);
   }

/* Check source line specifications for RECORD reference */

   if (frame_ref == RECORD)
   {
```

```
/* The first line must be in the data set */

      if ((long)sline_start < first_rec || (long)sline_start > last_rec)
      {
         error_handler(4);
         return(-1);
      }

/* The last line must be in the data set */

      if ((long)sline_end < first_rec || (long)sline_end > last_rec)
      {
         error_handler(5);
         return(-1);
      }
   }

/* Check source line specifications for DEPTH reference */

   else
   {

/* The first depth must be in the data set */

      if ((float)sline_start < last_depth || (float)sline_start > first_depth)
      {
         error_handler(6);
         return(-1);
      }

/* The last depth must be in the data set */

      if ((float)sline_end < last_depth || (float)sline_end > first_depth)
      {
         error_handler(7);
         return(-1);
      }

/* The first depth must be smaller than the last depth */

      if (selem_start > selem_end)
      {
         error_handler(8);
         return(-1);
      }
   }

/* Check display specification for channel 0 */

   if (disp_chan == CHAN_0)
   {
```

```
/* All the lines to display must fit */

      if ((dc0_line_start + sline_count - 1) > NO_DISP_LINES)
      {
         error_handler(9);
         return(-1);
      }

/* All the elements to display must fit */

      if (data_type == BOTH)
      {
         if ((dc0_elem_start + (selem_count * 2)) > NO_DISP_ELEMS)
         {
            error_handler(10);
            return(-1);
         }
      }

      else
      {
         if ((dc0_elem_start + selem_count - 1) > NO_DISP_ELEMS)
         {
            error_handler(11);
            return(-1);
         }
      }

   }

/* Check display specification for channel 1 */

   else
   {

/* All the lines to display must fit */

      if ((dc1_line_start + sline_count - 1) > NO_DISP_LINES)
      {
         error_handler(9);
         return(-1);
      }

/* All the elements to display must fit */

      if (data_type == BOTH)
      {
         if ((dc1_elem_start + (selem_count * 2)) > NO_DISP_ELEMS)
         {
            error_handler(10);
            return(-1);
         }
      }
```

```
      else
      {
         if ((dcl_elem_start + selem_count - 1) > NO_DISP_ELEMS)
         {
            error_handler(11);
            return(-1);
         }
      }
   }

/* All tests have been passed */

   return(0);
}



/*
*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*
*
*  MODULE NAME:  error_handler
*
*  DESCRIPTION:  error_handler was written to provide one central function
*    to handle the output of error messages.  This covers not only errors
*    detected by local functions but also errors returned by MSC functions.
*    This function is responsible for setting, writing an error message to,
*    and unsetting the error message window when called.
*
*  NOTES:  Passed parameter
*             index - specifies which error processing procedure to use
*                     (int)
*          Returned value - none
*
*    Error messages are allocated as follows:
*       Message number          Used by function
*           1 - 19                 check_parms
*          20 - 29                 alev2_opt11
*          30 - 39                 depth_rec_sel
*          40 - 49                 rec_sel_rec1
*          50 - 59                 rec_sel_rec2
*          60 - 69                 disp_recs
*
*    Not all the message numbers allocated are actually used.  Passing a
*    nonimplemented message number (index) will result in the error
*    message window being displayed with an error_handler error message.
*
*  DEVELOPED BY: Troy K. Moore
*  DATE:         10/7/87
*
*  REVISION NOTES:
*
*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*
*/
```

```
void  error_handler(index)

int   index;

{
   char  *msg1_1 = "  Initial element selected out of bounds, must be as follows:\n";
   char  *msg1_2 = "              1 <= first element <= %d";
   char  *msg2_1 = "   Final element selected out of bounds, must be as follows:\n";
   char  *msg2_2 = "              1 <= ending element <= %d";
   char  *msg3_1 = "   Element start and end in wrong order, must be as follows:\n";
   char  *msg3_2 = "              first element <= ending element";
   char  *msg4_1 = "  Initial record selected out of bounds, must be as follows:\n";
   char  *msg4_2 = "              %ld <= first record <= %ld";
   char  *msg5_1 = "   Final record selected out of bounds, must be as follows:\n";
   char  *msg5_2 = "              %ld <= ending record <= %ld";
   char  *msg6_1 = "  Initial depth selected out of bounds, must be as follows:\n";
   char  *msg6_2 = "              %6.1f <= first depth <= %6.1f";
   char  *msg7_1 = "  Final depth selected out of bounds, must be as follows:\n";
   char  *msg7_2 = "              %6.1f <= ending depth <= %6.1f";
   char  *msg8_1 = "  Start and end depth in wrong order, must be as follows:\n";
   char  *msg8_2 = "              start depth <= ending depth";
   char  *msg9_1 = "     Too many lines to display, must be as follows:\n";
   char  *msg9_2 = "  start display line + no lines to display - 1 <= %d";
   char *msg10_1 = "   Display line length is too long, must be as follows:\n";
   char *msg10_2 = "  start display line + (2 * no lines to display) <= %d";
   char *msg11_1 = "   Display line length is too long, must be as follows:\n";
   char *msg11_2 = "  start display line + no lines to display - 1 <= %d";

   char *msg20_1 = "            alev2_opt11: error opening data file\n";
   char *msg21_1 = "             alev2_opt11: error opening dp file\n";
   char *msg22_1 = "          alev2_opt11: error determining file length\n";
   char *msg23_1 = "          alev2_opt11: error during initial lseek\n";
   char *msg24_1 = "     alev2_opt11: error during initial read of dp file\n";
   char *msg25_1 = "          alev2_opt11: error during second lseek\n";
   char *msg26_1 = "      alev2_opt11: error during second read of dp file\n";

   char *msg30_1 = "    depth_sel_rec: error initial positioning file pointer\n";
   char *msg31_1 = "            depth_sel_rec: error reading dp file\n";
   char *msg32_1 = "       depth_sel_rec: error positioning file pointer\n";
   char *msg33_1 = "      depth_sel_rec: end of data reached prematurely\n";

   char *msg40_1 = "          rec_sel_rec1: error positioning file pointer\n";
   char *msg41_1 = "          rec_sel_rec1: end of data reached prematurely\n";

   char *msg50_1 = "          rec_sel_rec2: error positioning file pointer\n";
   char *msg51_1 = "          rec_sel_rec2: end of data reached prematurely\n";

   char *msg60_1 = "    disp_recs: error positioning file pointer in data file\n";
   char *msg61_1 = "             disp_recs: error reading data file\n";

   char *errno_line = "                         errno = %2d";

   char *def_line = "            error_handler: error %d not implemented";
```

```
   int   temp;

   extern int  errno;


/* Prepare for writing to the error message window */

   set_wn(error_wind);
   mv_cs(1,0,error_wind);

/* Select which message to output based on index */

   switch (index)
   {
      case 1:
         v_st(msg1_1,error_wind);
         v_printf(error_wind,msg1_2,NO_DATA_VALS);
         break;

      case 2:
         v_st(msg2_1,error_wind);
         v_printf(error_wind,msg2_2,NO_DATA_VALS);
         break;

      case 3:
         v_st(msg3_1,error_wind);
         v_st(msg3_2,error_wind);
         break;

      case 4:
         v_st(msg4_1,error_wind);
         v_printf(error_wind,msg4_2,first_rec,last_rec);
         break;

      case 5:
         v_st(msg5_1,error_wind);
         v_printf(error_wind,msg5_2,first_rec,last_rec);
         break;

      case 6:
         v_st(msg6_1,error_wind);
         v_printf(error_wind,msg6_2,last_depth,first_depth);
         break;

      case 7:
         v_st(msg7_1,error_wind);
         v_printf(error_wind,msg7_2,last_depth,first_depth);
         break;

      case 8:
         v_st(msg8_1,error_wind);
         v_st(msg8_2,error_wind);
         break;
```

```
case 9:
   v_st(msg9_1,error_wind);
   v_printf(error_wind,msg9_2,NO_DISP_LINES);
   break;

case 10:
   v_st(msg10_1,error_wind);
   v_printf(error_wind,msg10_2,NO_DISP_LINES);
   break;

case 11:
   v_st(msg11_1,error_wind);
   v_printf(error_wind,msg11_1,NO_DISP_LINES);
   break;

case 20:
   v_st(msg20_1,error_wind);
   v_printf(error_wind,errno_line,errno);
   break;

case 21:
   v_st(msg21_1,error_wind);
   v_printf(error_wind,errno_line,errno);
   break;

case 22:
   v_st(msg22_1,error_wind);
   v_printf(error_wind,errno_line,errno);
   break;

case 23:
   v_st(msg23_1,error_wind);
   v_printf(error_wind,errno_line,errno);
   break;

case 24:
   v_st(msg24_1,error_wind);
   v_printf(error_wind,errno_line,errno);
   break;

case 25:
   v_st(msg25_1,error_wind);
   v_printf(error_wind,errno_line,errno);
   break;

case 26:
   v_st(msg26_1,error_wind);
   v_printf(error_wind,errno_line,errno);
   break;
```

```
case 30:
   v_st(msg30_1,error_wind);
   v_printf(error_wind,errno_line,errno);
   break;

case 31:
   v_st(msg31_1,error_wind);
   v_printf(error_wind,errno_line,errno);
   break;

case 32:
   v_st(msg32_1,error_wind);
   v_printf(error_wind,errno_line,errno);
   break;

case 33:
   v_st(msg33_1,error_wind);
   break;

case 40:
   v_st(msg40_1,error_wind);
   v_printf(error_wind,errno_line,errno);
   break;

case 41:
   v_st(msg41_1,error_wind);
   break;

case 50:
   v_st(msg50_1,error_wind);
   v_printf(error_wind,errno_line,errno);
   break;

case 51:
   v_st(msg51_1,error_wind);
   break;

case 60:
   v_st(msg60_1,error_wind);
   v_printf(error_wind,errno_line,errno);
   break;

case 61:
   v_st(msg61_1,error_wind);
   v_printf(error_wind,errno_line,errno);
   break;

default:
   v_printf(error_wind,def_line,index);

}
```

```
/* Wait for input from the keyboard before removing window and continuing */

   temp = ki();
   unset_wn(error_wind);

   return;

}


/*
*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*
*
*   MODULE NAME:  recs_specs
*
*   DESCRIPTION:  This function is designed to show the user, via an
*     information window, the relationship of the records used to the
*     depth (or vice versa).  This function is called after the records
*     are actually displayed.
*
*   NOTES:  Passed parameters
*     fdpth - depth of the first record specified by the user, used only
*             when using RECORD reference (float).
*     ldpth - depth of the last record specified by the user, used only
*             when using RECORD reference (float).
*     frec  - record corresponding to the first depth specified by the
*             user, used only when using DEPTH reference (long int).
*     lrec  - record corresponding to the last depth specified by the
*             user, used only when using DEPTH reference (long int).
*
*   DEVELOPED BY: Troy K. Moore
*   DATE:         10/7/87
*
*   REVISION NOTES:
*
*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*
*/

void  recs_specs(fdpth,ldpth,frec,lrec)

long int    frec, lrec;
float       fdpth, ldpth;

{

   int      temp;



/* Prepare for writing to the rinfo window */

   set_wn(rinfo_wind);
   mv_cs(1,0,rinfo_wind);
```

```
/* If DEPTH reference, output record information */

   if (frame_ref == DEPTH)
   {
      v_printf(rinfo_wind,"Starting depth %5d -> Record %ld",sline_start,frec);
      mv_cs(2,0,rinfo_wind);
      v_printf(rinfo_wind,"Final depth    %5d -> Record %ld",sline_end,lrec);
   }

/* If RECORD reference, output depth information */

   else
   {
      v_printf(rinfo_wind,"Starting record %6d -> Depth %7.1f",sline_start,fdpth);
      mv_cs(2,0,rinfo_wind);
      v_printf(rinfo_wind,"Final record    %6d -> Depth %7.1f",sline_end,ldpth);
   }

/* Wait for input from the keyboard before removing window and continuing */

   temp = ki();
   unset_wn(rinfo_wind);

   return;

}


--------------------------------------------------------------------------
                            File: BATDACTS.C
--------------------------------------------------------------------------


/*
*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*
*
*  MODULE NAME:  batdacts
*
*  DESCRIPTION:  This file contains numerous functions that are called as
*    a direct result of a menu item being selected.  A function then may
*    call up a form to obtain numeric parameters from the user.  Each
*    function is responsible for updating the correct line(s) in the
*    status window.
*
*  NOTES:  Functions in this file are action functions defined in
*    batdmenu.c
*
*  DEVELOPED BY: Troy K. Moore
*  DATE:         10/5/87
*
```

Appendix J - BATD Source Code

```
*   REVISION NOTES:
*     12/7/87  TKM   Changed name of DT-IRIS include file.
*
*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*
*/

#define  WN_DEBUG

#include <stdlib.h>
#include <io.h>
#include <fcntl.h>

#include <batdxtrn.h>
#include <batdxwnd.h>
#include <batddefs.h>

#include <\image\dt_iris\isdefs.c>

/*>>>>>>>>>>>>>>>>>>>  alev_2_opt11 - Input file processing   <<<<<<<<<<<<<<<<*/

int   alev2_opt11(parm)
char *parm;

{
   int         read_flag;

   long int    seek_flag;

   struct rec_info   temp_buf;

   void     stat_wind_wrt(), error_handler();
   long int lseek();


/* Obtain filenames */

   fm_proc(0,fn_form);

/* If files are open, close them */

   if (dfh != 0)
      close(dfh);

   dfh = open(data_fname,O_RDONLY|O_BINARY);
   if (dfh == -1)
   {
      error_handler(20);
      return(-1);
   }

   if (dpfh != 0)
      close(dpfh);
```

```
   dpfh = open(dp_fname,O_RDONLY|O_BINARY);
   if (dpfh == -1)
   {
      error_handler(21);
      return(-1);
   }

/* Place the file names in the status window */

   stat_wind_wrt(2);

/* Determine the number of records in the input file */

   seek_flag = filelength(dpfh);
   if (seek_flag == -1L)
   {
      error_handler(22);
      return(-1);
   }

   last_rec = seek_flag/sizeof(struct rec_info);

/* Obtain the first last depths in the input file */

   seek_flag = lseek(dpfh,0L,SEEK_SET);
   if (seek_flag == -1L)
   {
      error_handler(23);
      return(-1);
   }

   read_flag = read(dpfh,(char *)&temp_buf,(unsigned int)sizeof(struct rec_info));
   if (read_flag < 0)
   {
      error_handler(24);
      return(-1);
   }

   first_depth = temp_buf.depth;

   seek_flag = lseek(dpfh,-1L*(sizeof(struct rec_info)),SEEK_END);
   if (seek_flag == -1L)
   {
      error_handler(25);
      return(-1);
   }

   read_flag = read(dpfh,(char *)&temp_buf,(unsigned int)sizeof(struct rec_info));
   if (read_flag < 0)
   {
      error_handler(26);
      return(-1);
   }
```

```
   last_depth = temp_buf.depth;

/* Place the first and last record & depth information in the status window */

   stat_wind_wrt(3);

   return(0);

}



/*>>>>>>>>>>>>>>>>>>>  alev3_opt121 - Select amplitude data  <<<<<<<<<<<<<<<<<<<*/

int   alev3_opt121(parm)
char *parm;

{
   void stat_wind_wrt();


/* User selects amplitude data for display */

   data_type = (int)AMPLITUDE;

/* Obtain both scale factors even though only one will be used */

   temp1 = amp_sf;
   temp2 = tt_sf;

   fm_proc(0,sf_form);

   amp_sf = temp1;
   tt_sf = temp2;

/* Place scale factors in the status window */

   stat_wind_wrt(4);

   return(1);

}


/*>>>>>>>>>>>>>>>>>>  alev3_opt122 - Select travel time data  <<<<<<<<<<<<<<<<<<*/

int   alev3_opt122(parm)
char *parm;

{
   void stat_wind_wrt();
```

```
/* User selects travel time data for display */

   data_type = (int)T_TIME;

/* Obtain both scale factors even though only one will be used */

   temp1 = amp_sf;
   temp2 = tt_sf;

   fm_proc(0,sf_form);

   amp_sf = temp1;
   tt_sf = temp2;

   stat_wind_wrt(4);

   return(1);

}



/*>>>>>>>>>>>>>>>>>>  alev3_opt123 - Select both data types  <<<<<<<<<<<<<<<<<*/

int   alev3_opt123(parm)
char *parm;

{
   void stat_wind_wrt();


/* User selects amplitude and travel time data for display */

   data_type = (int)BOTH;

/* Obtain both scale factors */

   temp1 = amp_sf;
   temp2 = tt_sf;

   fm_proc(0,sf_form);

   amp_sf = temp1;
   tt_sf = temp2;

/* Place scale factors in the status window */

   stat_wind_wrt(4);

   return(1);

}
```

```
/*>>>>>>>>>>>>>>>>>>>>>> alev3_opt132 - Select scan mode <<<<<<<<<<<<<<<<<<<<<*/

int   alev3_opt132(parm)
char *parm;

{
   void stat_wind_wrt();


/* User selects scan mode of operation */

   disp_mode = SCAN_MODE;

/* Update the status window */

   stat_wind_wrt(5);

   return(1);

}



/*>>>> alev5_opt13111 - Obtain source line start, count, inc (frame)  <<<<<*/

int   alev5_opt13111(parm)
char *parm;

{
   void stat_wind_wrt();


/* To reach this function, the user had to select frame mode of operation */

   disp_mode = FRAME_MODE;

/* Obtain the starting line, increment, and count information */

   temp1 = sline_start;
   temp2 = sline_inc;
   temp3 = sline_count;

   fm_proc(0,cnt_form);

   sline_start = temp1;
   sline_inc = temp2;
   sline_count = temp3;

/* Using this information, calculate the end value */

   sline_end = sline_start + (sline_inc * (sline_count - 1));
```

```
/* Update the status window */

   stat_wind_wrt(5);
   stat_wind_wrt(7);

   return(1);

}



/*>>>>>>  alev5_opt13112 - Obtain source line start, end, inc (frame)  <<<<<*/

int    alev5_opt13112(parm)
char *parm;

{
   void stat_wind_wrt();



/* To reach this function, the user had to select frame mode of operation */

   disp_mode = FRAME_MODE;

/* Obtain the starting line, increment, and end information */

   temp1 = sline_start;
   temp2 = sline_inc;
   temp3 = sline_end;

   fm_proc(0,end_form);

   sline_start = temp1;
   sline_inc = temp2;
   sline_end = temp3;

/* Using this information, calculate the count value, updating end if req'd */

   sline_count = ((sline_end - sline_start)/sline_inc) + 1;
   sline_end = sline_start + (sline_inc * (sline_count - 1));

/* Update the status window */

   stat_wind_wrt(5);
   stat_wind_wrt(7);

   return(1);

}
```

```
/*>>>  alev5_opt13121 - Obtain source elememt start, count, inc (frame)  <<<*/

int   alev5_opt13121(parm)
char *parm;

{
   void stat_wind_wrt();


/* To reach this function, the user had to select frame mode of operation */

   disp_mode = FRAME_MODE;

/* Obtain the starting element, increment, and count information */

   temp1 = selem_start;
   temp2 = selem_inc;
   temp3 = selem_count;

   fm_proc(0,cnt_form);

   selem_start = temp1;
   selem_inc = temp2;
   selem_count = temp3;

/* Using this information, calculate the end value */

   selem_end = selem_start + (selem_inc * (selem_count - 1));

/* Update the status window */

   stat_wind_wrt(5);
   stat_wind_wrt(8);

   return(1);

}


/*>>>>  alev5_opt13122 - Obtain source element start, end, inc (frame)  <<<<*/

int   alev5_opt13122(parm)
char *parm;

{
   void stat_wind_wrt();



/* To reach this function, the user had to select frame mode of operation */

   disp_mode = FRAME_MODE;
```

```
/* Obtain the starting element, increment, and end information */

   temp1 = selem_start;
   temp2 = selem_inc;
   temp3 = selem_end;

   fm_proc(0,end_form);

   selem_start = temp1;
   selem_inc = temp2;
   selem_end = temp3;

/* Using this information, calculate the count value, updating end if req'd */

   selem_count = ((selem_end - selem_start)/selem_inc) + 1;
   selem_end = selem_start + (selem_inc * (selem_count - 1));

/* Update the status window */

   stat_wind_wrt(5);
   stat_wind_wrt(8);

   return(1);

}



/*>>>>>>>>>>>  alev5_opt13131 - Select depth reference (frame)  <<<<<<<<<<<<<*/

int    alev5_opt13131(parm)
char *parm;

{
   void stat_wind_wrt();



/* To reach this function, the user had to select frame mode of operation */

   disp_mode = (int)FRAME_MODE;
   frame_ref = (int)DEPTH;

/* Update the status window */

   stat_wind_wrt(5);

   return(2);

}
```

```
/*>>>>>>>>>>  alev5_opt13132 - Select record reference (frame)  <<<<<<<<<<<<<*/

int   alev5_opt13132(parm)
char *parm;

{
   void stat_wind_wrt();



/* To reach this function, the user had to select frame mode of operation */

   disp_mode = (int)FRAME_MODE;
   frame_ref = (int)RECORD;

/* Update the status window */

   stat_wind_wrt(5);

   return(2);

}


/*>>>>>>>>>>>>>>>>>  alev2_opt22 - Location parms for chan 0  <<<<<<<<<<<<<<<<*/

int   alev2_opt22(parm)
char  *parm;

{
   void stat_wind_wrt();



/* Obtain starting line and element location */

   temp1 = dc0_line_start;
   temp2 = dc0_elem_start;

   fm_proc(0,disp_form);

   dc0_line_start = temp1;
   dc0_elem_start = temp2;

/* Update the status window */

   stat_wind_wrt(7);

   return(1);

}
```

```
/*>>>>>>>>>>>>>>>>>  alev2_opt23 - Location parms for chan 1  <<<<<<<<<<<<<<<<*/

int   alev2_opt23(parm)
char  *parm;

{
   void stat_wind_wrt();



/* Obtain starting line and element location */

   temp1 = dc1_line_start;
   temp2 = dc1_elem_start;

   fm_proc(0,disp_form);

   dc1_line_start = temp1;
   dc1_elem_start = temp2;

/* Update the status window */

   stat_wind_wrt(8);

   return(1);

}



/*>>>>>>>>>>>>>>>>>>>>>  alev3_opt211 - Select channel 0  <<<<<<<<<<<<<<<<<<<<*/

int   alev3_opt211(parm)
char  *parm;

{
   void stat_wind_wrt();



/* Make channel 0 the displayed channel */

   disp_chan = CHAN_0;
   is_select_output_frame(disp_chan);

/* Update the status window */

   stat_wind_wrt(6);

   return(1);

}
```

```
/*>>>>>>>>>>>>>>>>>>>>>  alev3_opt212 - Select channel 1  <<<<<<<<<<<<<<<<<<<<*/

int   alev3_opt212(parm)
char  *parm;

{
   void stat_wind_wrt();



/* Make channel 0 the displayed channel */

   disp_chan = CHAN_1;
   is_select_output_frame(disp_chan);

/* Update the status window */

   stat_wind_wrt(6);

   return(1);

}



/*>>>>>>>>>>>>>>>>>>>>>>>>>>  arun - Display data  <<<<<<<<<<<<<<<<<<<<<<<<<<*/

int   arun(parm)
char *parm;

{
   int   ret_val;
   int   check_parms();



/* Check the parameters for validity */

   ret_val = check_parms();

/* Select return procedure based on check_parms findings */

   ret_val = (ret_val == 0) ? -2 : -1;

   return(ret_val);

}
```

```
/*>>>>>>>>>>>>>>>>>>>>>>>>>> exit - Exit algorithm  <<<<<<<<<<<<<<<<<<<<<<<<<<*/

int   aexit(parm)
char *parm;

{

/* Change the run_flag */

   run_flag = STOP;

   return(-2);

}


==========================================================================
                          File: BATDRSEL.C
==========================================================================


/*
*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*
*
*   MODULE NAME:  depth_sel_rec
*
*   DESCRIPTION:  This module is designed to select records from the data
*     file that most closely match the depth specifications given by the
*     user.  Record locations and lengths are determined using the dp file
*     and returned to the calling routine via two buffers.  The record
*     numbers of the first and last records used are returned to the calling
*     routine.  Records are selected based on the absolute value of the
*     difference between the record's depth and a calculated ideal next
*     record depth.
*
*   NOTES:  Passed parameters
*              loc_buf - buffer to contain long int offsets of rec selected
*              len_buf - buffer to contain int lengths of rec selected
*              frec    - record number corresponding to sline_start depth
*              lrec    - record number corresponding to sline_end depth
*           Return values
*               0 - no error detected
*              -1 - error detected
*
*   DEVELOPED BY: Troy K. Moore
*   DATE:         10/6/87
*
*   REVISION NOTES:
*     12/4/87  TKM   Added code to update cur_dif and next_dif with respect
*                    to the new target depth after a depth match occurred.
*     12/7/87  TKM   Modified debug output to send it to stdprn.
*
*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*
*/
```

Appendix J - BATD Source Code

```
#include    <io.h>
#include    <stdio.h>
#include    <batdxtrn.h>
#include    <batddefs.h>

static struct rec_info     info_buf[NO_RECS_PROC];

int   depth_sel_rec(loc_buf,len_buf,frec,lrec)

int         len_buf[];
long int    loc_buf[], *frec, *lrec;

{
   int         found_count, read_bytes, target_depth, count;

   long int    loop_count, seek_flag;

   float       cur_dif, prev_dif, next_dif;

   void        error_handler();

   long int    lseek();


   #ifdef DEBUG_PRINT
      fprintf(stdprn,"entering depth_sel_rec\n");
      fprintf(stdprn,"sline_start: %d sline_count: %d\n",sline_start,sline_count);
      fprintf(stdprn,"sline_inc: %d sline_end: %d\n\n",sline_inc,sline_end);
   #endif

/* Initialize variables */

   found_count = sline_count - 1;
   target_depth = sline_end;
   loop_count = 0L;

/* Set the file pointer to the start of the dp file */

   seek_flag = lseek(dpfh,0L,SEEK_SET);
   if (seek_flag == -1L)
   {
      error_handler(30);
      return(-1);
   }

/* Pre-load buffer */

   read_bytes = read(dpfh,(char *)info_buf,(unsigned int)sizeof(info_buf));
   if (read_bytes < 0)
   {
      error_handler(31);
      return(-1);
   }
```

```
/* Calculate first current difference and set up first previous difference as
   loop only calculates new next difference */

   cur_dif = info_buf[0].depth - target_depth;
   cur_dif = (cur_dif < 0.0) ? -1.0 * cur_dif : cur_dif;

   prev_dif = 10000.0;

/* Loop until EOF encountered */

   while (read_bytes > 0)
   {

/* Process each depth in the buffer */

      for (count = 0; count < ((read_bytes/sizeof(struct rec_info)) - 1);
            count++)
      {
         next_dif = info_buf[count+1].depth - target_depth;
         next_dif = (next_dif < 0.0) ? -1.0 * next_dif : next_dif;


/* Depth match */

         if (cur_dif < prev_dif && cur_dif < next_dif && cur_dif < CHECK_BAND)
         {
            if (found_count == sline_count - 1)
               *lrec = loop_count * (NO_RECS_PROC - 1) + count + 1;

            loc_buf[found_count] = info_buf[count].rec_offset;
            len_buf[found_count] = info_buf[count].rec_length;


            #ifdef DEBUG_PRINT
               fprintf(stdprn,"found_count: %d depth: %.2f\n",found_count,
                     info_buf[count].depth);
               fprintf(stdprn,"offset: %lx length: %d\n\n",loc_buf[found_count],
                     len_buf[found_count]);
            #endif

            target_depth = target_depth - sline_inc;  /* calc next depth */

            if (found_count == 0)             /* last depth has been found */
            {
               *frec = loop_count * (NO_RECS_PROC - 1) + count;
               return(0);                  /* errorless return */
            }

            found_count--;
```

```
/* Update differences as they now refer to the new target depth */

         cur_dif = cur_dif + sline_inc;
         next_dif = next_dif + sline_inc;

         }


/* Update differences in preparation for next check */

         prev_dif = cur_dif;
         cur_dif = next_dif;

      }

/* Reposition file pointer for next read */

      seek_flag = lseek(dpfh,(long int)-sizeof(struct rec_info),SEEK_CUR);
      if (seek_flag == -1L)
      {
         error_handler(32);
         return(-1);
      }

/* Read to fill buffer */

      read_bytes = read(dpfh,(char *)info_buf,(unsigned int)sizeof(info_buf));
      loop_count++;

   }

/* This code reached only if ran out of data before all records selected */

   error_handler(33);
   return(-1);

}


/*
*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*
*
*  MODULE NAME:  rec_sel_recl
*
*  DESCRIPTION:  This module is designed to select records in the RECORD
*     mode that correspond to those selected by the user.  Record locations
*     and lengths are determined using the dp file and returned to the
*     calling routine via two buffers.  The depths of the first and last
*     record selected are returned to the calling routine.  This function
*     should only be entered when sline_int is negative.
*
```

```
*   NOTES:  Passed parameters
*              loc_buf - buffer to contain long int offsets of rec selected
*              len_buf - buffer to contain int lengths of rec selected
*              fdepth  - depth corresponding to record sline_start
*              ldepth  - depth corresponding to record sline_end
*           Return values
*               0 - no error detected
*              -1 - error detected
*
*   DEVELOPED BY: Troy K. Moore
*   DATE:         10/6/87
*
*   REVISION NOTES:
*     12/7/87  TKM   Modified debug output to send it to stdprn.
*
*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*
*/

int   rec_sel_rec1(loc_buf,len_buf,fdepth,ldepth)

int         len_buf[];
long int    loc_buf[];
float       *fdepth, *ldepth;

{
   int         rbuf_point, iobuf_point, read_bytes;

   long int    seek_flag, loop_count;

   void        error_handler();
   long int    lseek();



/* Position file pointer to first record to process which, in this case, is
   the record specified by sline_end */

   seek_flag = lseek(dpfh,(long int)(sizeof(struct rec_info)*(sline_end-1)),
         SEEK_SET);
   if (seek_flag == -1L)
   {
      error_handler(40);
      return(-1);
   }

/* Initialize pointer into return buffer */

   rbuf_point = sline_count - 1;

/* Preload the buffer */

   read_bytes = read(dpfh,(char *)info_buf,(unsigned int)sizeof(info_buf));
```

```
/* Loop until EOF encountered */

   for (iobuf_point = 0, loop_count = 0L; read_bytes > 0; loop_count++)
   {

/* Extract only selected records from buffer */

      for ( ; iobuf_point < (read_bytes/sizeof(struct rec_info));
            iobuf_point -= sline_inc)
      {

         loc_buf[rbuf_point] = info_buf[iobuf_point].rec_offset;
         len_buf[rbuf_point] = info_buf[iobuf_point].rec_length;

         #ifdef DEBUG_PRINT
            fprintf(stdprn,"loop_count: %ld rbuf_point: %d iobuf_point: %d\n",
                  loop_count,rbuf_point,iobuf_point);
            fprintf(stdprn,"offset: %lx length: %d depth: %.2f\n\n",
                  loc_buf[rbuf_point],len_buf[rbuf_point],
                  info_buf[iobuf_point].depth);
         #endif

         if (iobuf_point == 0 && loop_count == 0L)
            *ldepth = info_buf[iobuf_point].depth;

         if (rbuf_point == 0)             /* last record has been found */
         {
            *fdepth = info_buf[iobuf_point].depth;
            return(0);                    /* errorless return */
         }

         rbuf_point--;
      }

/* Reposition pointer into input buffer for next read */

      iobuf_point = iobuf_point - NO_RECS_PROC;

/* Read to re-fill the input buffer */

      read_bytes = read(dpfh,(char *)info_buf,(unsigned int)sizeof(info_buf));

   }

/* This code reached only if ran out of data before all records selected */

   error_handler(41);
   return(-1);


}
```

Appendix J - BATD Source Code

```
/*
*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*
*
*  MODULE NAME:  rec_sel_rec2
*
*  DESCRIPTION:  This module is designed to select records in the RECORD
*    mode that correspond to those selected by the user.  Record locations
*    and lengths are determined using the dp file and returned to the
*    calling routine via two buffers.  The depths of the first and last
*    record selected are returned to the calling routine.  This function
*    should only be entered when sline_int is positive.
*
*  NOTES:  Passed parameters
*             loc_buf - buffer to contain long int offsets of rec selected
*             len_buf - buffer to contain int lengths of rec selected
*             fdepth  - depth corresponding to record sline_start
*             ldepth  - depth corresponding to record sline_end
*          Return values
*              0 - no error detected
*             -1 - error detected
*
*  DEVELOPED BY: Troy K. Moore
*  DATE:         10/6/87
*
*  REVISION NOTES:
*    12/7/87  TKM   Modified debug output to send it to stdprn.
*
*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*
*/

int   rec_sel_rec2(loc_buf,len_buf,fdepth,ldepth)

int         len_buf[];
long int    loc_buf[];
float       *fdepth, *ldepth;

{
   int         rbuf_point, iobuf_point, read_bytes;

   long int    seek_flag, loop_count;

   void        error_handler();
   long int    lseek();


/* Position file pointer to first record to process which, in this case, is
   the record specified by sline_start */

   seek_flag = lseek(dpfh,(long int)(sizeof(struct rec_info)*(sline_start-1)),SEEK_SET);
```

```
    if (seek_flag == -1L)
    {
        error_handler(50);
        return(-1);
    }

/* Initialize pointer into return buffer */

    rbuf_point = 0;

/* Preload the buffer */

    read_bytes = read(dpfh,(char *)info_buf,(unsigned int)sizeof(info_buf));

/* Loop until EOF encountered */

    for (iobuf_point = 0, loop_count = 0L; read_bytes > 0; loop_count++)
    {

/* Extract only selected records from buffer */

        for ( ; iobuf_point < (read_bytes/sizeof(struct rec_info));
                iobuf_point += sline_inc)
        {

            loc_buf[rbuf_point] = info_buf[iobuf_point].rec_offset;
            len_buf[rbuf_point] = info_buf[iobuf_point].rec_length;

            #ifdef DEBUG_PRINT
                fprintf(stdprn,"loop_count: %ld rbuf_point: %d iobuf_point: %d\n",
                        loop_count,rbuf_point,iobuf_point);
                fprintf(stdprn,"offset: %lx length: %d depth: %.2f\n",
                        loc_buf[rbuf_point],len_buf[rbuf_point],
                        info_buf[iobuf_point].depth);
            #endif

            if (iobuf_point == 0 && loop_count == 0L)
                *fdepth = info_buf[iobuf_point].depth;

            if (rbuf_point == sline_count - 1)  /* last record has been found */
            {
                *ldepth = info_buf[iobuf_point].depth;
                return(0);                      /* errorless return */
            }

            rbuf_point++;
        }

/* Reposition pointer into input buffer for next read */

        iobuf_point = iobuf_point - NO_RECS_PROC;
```

```
/* Read to re-fill the input buffer */

      read_bytes = read(dpfh,(char *)info_buf,(unsigned int)sizeof(info_buf));

   }

/* This code reached only if ran out of data before all records selected */

   if (rbuf_point < sline_count)
   {
      error_handler(51);
      return(-1);
   }

}



/*
*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*
*
*   MODULE NAME:  disp_recs
*
*   DESCRIPTION:  This function will take the two buffers of record
*     information generated by *_sel_rec functions and extract the user-
*     specified elements from the data records.  The extracted data will
*     be scaled and written to the specified location of the specified
*     frame buffer for display.  If both amplitude and travel time data
*     are requested, a 1 pixel void is left to separate the two data types.
*
*   NOTES:  Passed Parameters
*              loc_buf - buffer containing long int offsets of rec selected
*              len_buf - buffer containing int lengths of rec selected
*           Return values
*               0 - no error detected
*              -1 - error detected
*
*   DEVELOPED BY: Troy K. Moore
*   DATE:         10/6/87
*
*   REVISION NOTES:
*
*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*
*/

int   disp_recs(locbuf,lenbuf)

int      lenbuf[];
long int locbuf[];

{

   int         line_count, rbuf_point, dbuf_point;
   int         read_flag, amp_sfactor, tt_sfactor, start_line, start_elem;
```

```
    int         disp_pixels, proc_pixels, zero_count;

    long int    seek_flag;

    unsigned int      read_buf[NO_DATA_VALS * 2], disp_buf[NO_DISP_ELEMS];

    void        error_handler();
    long int    lseek();
    double      pow();


/* Initialize destination parameters */

    if (disp_chan == CHAN_0)
    {
       start_line = dc0_line_start;
       start_elem = dc0_elem_start;
    }

    else
    {
       start_line = dc1_line_start;
       start_elem = dc1_elem_start;
    }

/* Set up scale factors */

    amp_sfactor = pow(2.0,(double)amp_sf);
    tt_sfactor = pow(2.0,(double)tt_sf);

    disp_pixels = sizeof(disp_buf)/sizeof(int);

/* Begin main processing loop */

    for (line_count = 0; line_count < sline_count; line_count++)
    {

/* Position the file pointer within the data file */

       seek_flag = lseek(dfh,locbuf[line_count]+(long int)REV_HEAD_LEN,SEEK_SET);
       if (seek_flag == -1L)
       {
          error_handler(60);
          return(-1);
       }

/* Read in only the data portion of the record */

       read_flag = read(dfh,(char *)read_buf,
                         (unsigned int)(lenbuf[line_count]-REV_HEAD_LEN));
```

```
      if (read_flag < 0)
      {
         error_handler(61);
         return(-1);
      }

/* Zero the display buffer */

      for (zero_count = 0; zero_count < disp_pixels; zero_count++)
         disp_buf[zero_count] = 0;

/* Set up the pointer into the read buffer */

      rbuf_point = selem_start - 1;

/* Calculate the number of elements to process and display */

      proc_pixels = (lenbuf[line_count] < selem_count) ? lenbuf[line_count] :
            selem_count;
      disp_pixels = (data_type == BOTH) ? NO_DATA_VALS + proc_pixels + 1:
            proc_pixels;

/* Process all the selected elements in the read buffer */

      for (dbuf_point = 0; dbuf_point < proc_pixels; dbuf_point++,
            rbuf_point+=selem_inc)
      {

         if (data_type == AMPLITUDE)
            disp_buf[dbuf_point] = read_buf[rbuf_point]/amp_sfactor;

         else if (data_type == T_TIME)
            disp_buf[dbuf_point] = read_buf[rbuf_point+NO_DATA_VALS]/tt_sfactor;

         else
         {
            disp_buf[dbuf_point] = read_buf[rbuf_point]/amp_sfactor;
            disp_buf[dbuf_point+NO_DATA_VALS+1] =
                        read_buf[rbuf_point+NO_DATA_VALS]/tt_sfactor;
         }

      }

/* Display the line that has been assembled in the display buffer */

      is_put_pixel(disp_chan,line_count+start_line-1,start_elem-1,disp_pixels,
            disp_buf);

   }

/* Return to the calling routine */

   return(0);
}
```

Appendix K - BATT Source Code

```
========================================================================
                          File: BATDDEFS.H
========================================================================


#define  VER               1.00     /* software version number */
#define  VER_DATE          "11/13/87"
#define  DISP_LINES        480      /* max no of displayable lines */
#define  TRACES_DISP       128      /* max no of traces per display */
#define  MAX_DISPS         8        /* max no of successive displays */
                                    /* used only to calculate RECS_SEL */
#define  RECS_SEL          TRACES_DISP * MAX_DISPS    /* max no of traces */
                                    /* or recs to process */
#define  MIN_INC           .5       /* min increment that will be allowed */
#define  POINTS_PER_TRACE  128      /* data points per trace */
#define  NO_RECS_PROC      512      /* no of recs to process in determining */
                                    /* recs to use (select_recs) */
#define  CHECK_BAND        .5       /* used in select_recs to determine */
                                    /* if local min is close enough to */
                                    /* target depth to use */
#define  FNAME_LEN         40       /* length of file name buffers */
#define  REV_HEAD          48       /* no bytes in rev header */
#define  CON_VAL           508.0    /* conversion from caliper in mm X 10 */
                                    /* to radius in inches */
#define  PIXELS_PER_POINT  2        /* no of columns over which to display */
                                    /* each caliper value */
#define  START_COLUMN      140      /* display column to place first data */
                                    /* value */
#define  TRACE_INTENSITY   255      /* intensity to draw traces */
#define  TICK_LINE_INTENSITY  100   /* intensity to draw tick lines */
#define  TEXT_INTENSITY    150      /* intensity to print text */
#define  DISP_FRAME          0      /* frame buffer to use */

/* Annotation positioning parameters */

#define  DEPTH_COL         8        /* display column to start depth info */
#define  REF_COL           100      /* display column to start tick line info */
#define  MIN_COL           410      /* display column to start min val info */
#define  MAX_COL           472      /* display column to start max val info */

#define  HALF_CHAR         7        /* offset from center line to position */
                                    /* text */

/* Set up and name the structure templates */

typedef struct     {
                     long int    rec_offset;
                     float       depth;
                     int         rec_length;
                   }  REC_INFO;
```

```
typedef struct    {
                      float       depth;
                      long int    rec_no;
                      float       min;
                      float       max;
                  }   SEL_REC;

------------------------------------------------------------------------
                              File: BATTGLOB.H
------------------------------------------------------------------------

char        data_fname[FNAME_LEN], dp_fname[FNAME_LEN];

int         no_traces, no_disps, traces_per_disp;
int         dfh, dpfh, rsel_flag, run_flag;
int         tick_lines[TRACES_DISP+2][2], map_data[POINTS_PER_TRACE][2];

float       first_file_depth, last_file_depth, dlines_per_inch;

double      init_depth, fin_depth, inc_depth, base_tic, tic_res;

REC_INFO    info_buf, *info_buf_ptr;

SEL_REC     selected_recs[RECS_SEL];

------------------------------------------------------------------------
                              File: BATTWND.H
------------------------------------------------------------------------

extern char    data_fname[], dp_fname[];

extern int     no_traces, no_disps, traces_per_disp;
extern int     dfh, dpfh, rsel_flag, run_flag;
extern int     tick_lines[TRACES_DISP+2][2], map_data[POINTS_PER_TRACE][2];

extern float   first_file_depth, last_file_depth, dlines_per_inch;

extern double  init_depth, fin_depth, inc_depth, base_tic, tic_res;

extern REC_INFO   info_buf, *info_buf_ptr;

extern SEL_REC    selected_recs[];
```

Appendix K - BATT Source Code

```
=========================================================================
                              File: BATTXTRN.H
=========================================================================


extern char     data_fname[], dp_fname[];

extern int      no_traces, no_disps, traces_per_disp;
extern int      dfh, dpfh, rsel_flag, run_flag;
extern int      tick_lines[TRACES_DISP+2][2], map_data[POINTS_PER_TRACE][2];

extern float    first_file_depth, last_file_depth, dlines_per_inch;

extern double   init_depth, fin_depth, inc_depth, base_tic, tic_res;

extern REC_INFO   info_buf, *info_buf_ptr;

extern SEL_REC    selected_recs[];


=========================================================================
                              File: BATTXWND.H
=========================================================================


#include <wfd.h>

extern DFORMPTR       fn_form, rspec_form, dspec_form, menu_entry;

extern WINDOWPTR      stat_wind, info_wind, errorx_wind, title_wind;

extern WINDOW         status_wn, info_wn, errorx_wn, title_wn;


=========================================================================
                              File: BATD.C
=========================================================================

/*
*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*
*
*   MODULE NAME:  batt
*
*   DESCRIPTION:  This is the main module of the code designed to generate
*     radial traces from BAT tool travel time data.  Functions are used
*     extensively to modularize implementation of this task.
*
*   NOTES:  Travel time data are assumed to have been mapped into caliper
*     in mm X 10 when collected in the field.
*     This algorithm was developed in response to a request from Don
*     Dreesen.
*
```

Appendix K - BATT Source Code

```
*   DEVELOPED BY: Troy K. Moore
*   DATE:         11/10/87
*
*   REVISION NOTES:
*
*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*
*/

#define     F_FLOAT
#define     WN_DEBUG

#include    <battdefs.h>
#include    <battglob.h>
#include    <battwnd.h>
#include    <\image\dt_iris\isdefs.c>

main()

{

   int      ret_flag, disp_count, exit_val, cont_flag;

   void     init_parms(), calc_tick_lines(), put_title();
   void     window_setup(), dform_setup(), menu_setup(), fill_stat_wind();
   int      select_recs(), intrvl_calc(), gen_disp(), continue_disp();
   int      check_parms();


   init_wfd();                          /* initialize windows for data */
   is_initialize();                     /* initialize dt-iris */
   cls();                               /* clear the system monitor */
   is_select_olut(0);                   /* select output lut */
   is_display(1);                       /* enable the display of data */
   is_select_output_frame(DISP_FRAME);  /* select frame buffer */
   init_parms();                        /* initialize parameters */
   window_setup();                      /* set up the windows */
   dform_setup();                       /* create the data forms */
   menu_setup();                        /* create the menu structure */
   put_title();                         /* display the title block */
   set_wn(stat_wind);                   /* place the status window on the */
                                        /* monitor */
   fill_stat_wind();                    /* fill in information in the status */
                                        /* window */
   while (ret_flag != -1 && run_flag == 0)
   {
      ret_flag = mn_proc(0,menu_entry);    /* obtain parameters from user */
      if (run_flag == -1)
         break;

      ret_flag = check_parms();         /* check parameters for validity */
      if (ret_flag != 0)
         continue;
```

```
      if (rsel_flag != 0)             /* identify new records if necessary */
      {
         ret_flag = select_recs();    /* determine which records to use */
         if (ret_flag != 0)
            break;

         ret_flag = intrvl_calc();    /* calc min & max between traces */
         if (ret_flag != 0)
            break;
      }

      calc_tick_lines();              /* determine where tick lines should */
                                      /* be placed */

      for (disp_count = 0, cont_flag = 0;
            disp_count < no_disps && cont_flag == 0; disp_count++)
      {
         ret_flag = gen_disp(disp_count); /* map and display data */
         if (ret_flag != 0)
            break;

         ret_flag = continue_disp(disp_count); /* continue to next disp? */
         if (ret_flag != 0)
            cont_flag = -1;

         is_frame_clear(DISP_FRAME);  /* clear the display */
      }

      rsel_flag = 0;                  /* reset the record select flag */
   }

   unset_wn(stat_wind);               /* remove the status window */
   is_end();                          /* dt-iris cleanup routine */
   close(dpfh);                       /* close data files */
   close(dfh);
   exit_val = (ret_flag == 0) ? 0 : -1; /* determine exit value */
   exit(exit_val);
}

=========================================================================
                              File: BATTSTUP.C
=========================================================================

/*
*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*
*
*  MODULE NAME:  init_parms
*
*  DESCRIPTION:  This function initializes variables/arrays with start
*     up values for use in BATT.
*
*  NOTES:
```

Appendix K - BATT Source Code

```
*
*   DEVELOPED BY: Troy K. Moore
*   DATE:         11/10/87
*
*   REVISION NOTES:
*
*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*
*/

#define     F_FLOAT
#define     WN_DEBUG

#include    <battdefs.h>
#include    <battxtrn.h>
#include    <battxwnd.h>

void init_parms()

{

   int      count;


/* Initialize the display parameters */

   no_traces = 0;
   no_disps = 0;
   traces_per_disp = 4;

/* Initialize all the floating point depth parameters */

   init_depth = 0.0;
   fin_depth = 0.0;
   inc_depth = 1.0;

/* Initialize the display tick parameters */

   base_tic = 4.0;
   tic_res = 1.0;

/* Initialize the file handles to 0 to indicate files not yet opened */

   dfh = 0;
   dpfh = 0;

/* Clear the record select flag */

   rsel_flag = 0;
   run_flag = 0;
```

```
/* Initialize the column values for the end points of the tick lines */

   for (count = 0; count < (TRACES_DISP + 2); count++)
      tick_lines[count][1] = START_COLUMN + (POINTS_PER_TRACE *
                              PIXELS_PER_POINT);

/* Initialize the column values for the end points of the tick lines */

   for (count = 0; count < POINTS_PER_TRACE; count++)
      map_data[count][1] = START_COLUMN + (count * PIXELS_PER_POINT);

/* Set up the dp record buffer pointer */

   info_buf_ptr = &info_buf;

   return;

}


/*
*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*
*
*   MODULE NAME:  window_setup
*
*   DESCRIPTION:  This function defines all the windows used by BATT.
*
*   NOTES:
*
*   DEVELOPED BY: Troy K. Moore
*   DATE:         10/29/87
*
*   REVISION NOTES:
*
*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*
*/

void window_setup()

{

/* Configure the status window */

   stat_wind = &status_wn;
   defs_wn(stat_wind,13,0,12,80,BDR_LNP);
   sw_name(" Status Window ",stat_wind);
   sw_namelocation(BOTTOMCENTER,stat_wind);
```

```
/* Configure the error window */

   errorx_wind = &errorx_wn;
   defs_wn(errorx_wind,7,5,4,70,BDR_LNP);
   sw_popup(ON,errorx_wind);
   sw_att(LRED,errorx_wind);
   sw_name(" Strike any key to return to menu ",errorx_wind);
   sw_namelocation(BOTTOMCENTER,errorx_wind);

/* Provide a window for general information */

   info_wind = &info_wn;
   defs_wn(info_wind,7,5,4,70,BDR_LNP);
   sw_popup(ON,info_wind);
   sw_name(" Strike any key to continue ",info_wind);
   sw_namelocation(BOTTOMCENTER,info_wind);

/* Create a window for the title information */

   title_wind = &title_wn;
   defs_wn(title_wind,4,20,9,40,BDR_DLNP);
   sw_popup(ON,title_wind);
   sw_name(" Strike any key to begin ",title_wind);
   sw_namelocation(BOTTOMCENTER,title_wind);

/* Modify the message window associated with the menu structure */

   se_mnmsg(MANUAL);
   sw_border(BDR_0P,&mnu_msgw);
   mod_wn(12,0,1,80,&mnu_msgw);

   return;

}


/*
*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*
*
*  MODULE NAME:  dform_setup
*
*  DESCRIPTION:  This function is designed to define all the data forms
*    used by BATT.
*
*  NOTES:
*
*  DEVELOPED BY: Troy K. Moore
*  DATE:         11/2/87
*
*  REVISION NOTES:
*
*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*
*/
```

```
void dform_setup()

{

/* Modify the default parameters for all the data forms */

   se_fmopt(FMDATA,AUTOEXIT);

/* Create a data form to obtain the file names */

   fn_form = fm_def(6,24,4,56,LNORMAL,BDR_DLNP);
   fld_def(0,0,"Filename:     ",FADJACENT,def_pic('X',FNAME_LEN-1),F_STRING,
         data_fname,fn_form);
   fld_def(1,0,"DP filename: ",FADJACENT,def_pic('X',FNAME_LEN-1),F_STRING,
         dp_fname,fn_form);

/* Create a data form to obtain the record selection parameters */

   rspec_form = fm_def(6,25,5,40,LNORMAL,BDR_DLNP);
   fld_def(0,0,"Initial depth:  ",FADJACENT,"#####@##",F_FLOAT,
         (char *)&init_depth,rspec_form);
   fld_def(1,0,"Final depth:    ",FADJACENT,"#####@##",F_FLOAT,
         (char *)&fin_depth,rspec_form);
   fld_def(2,0,"Increment:      ",FADJACENT,"###@##",F_FLOAT,
         (char *)&inc_depth,rspec_form);

/* Create a data form to obtain the display/mapping parameters */

   dspec_form = fm_def(6,15,5,60,LNORMAL,BDR_DLNP);
   fld_def(0,0,"Traces per display:  ",FADJACENT,"999",F_INT,
         (char *)&traces_per_disp,dspec_form);
   fld_def(1,0,"Base tick line (inches):    ",FADJACENT,"##@##",F_FLOAT,
         (char *)&base_tic,dspec_form);
   fld_def(2,0,"Resolution between tick lines (inches):      ",FADJACENT,
         "##@##",F_FLOAT,(char *)&tic_res,dspec_form);

   return;

}

/*
*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*
*
*  MODULE NAME:  menu_setup
*
*  DESCRIPTION:  This function is designed to define the menu structure
*    used by BATT.
*
*  NOTES:
*
*  DEVELOPED BY: Troy K. Moore
*  DATE:         10/29/87
*
```

```
*   REVISION NOTES:
*     12/7/87  TKM   Added code to prevent user from exiting out the top
*                    of the menu via ESC.
*
*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*
*/

void menu_setup()

{

   char  *p_lev1_opt1    =  "Specify which traces to display";
   char  *p_lev1_opt2    =  "Display the selected traces";
   char  *p_lev1_opt3    =  "Exit to DOS";

   char  *p_lev2_opt11   =  "Specify input file names";
   char  *p_lev2_opt12   =  "Identify data to display";
   char  *p_lev2_opt13   =  "Specify display orientation of data";

   int   alev2_opt11(), alev2_opt12(), alev2_opt13();
   int   adisplay(), aexit();

   DFORMPTR    lev2_opt1;

   DFIELDPTR   mnf_def();
   DFORMPTR    mnfm_def(), mnfm_dap();


/* Configure the top level menu */

   menu_entry = mnfm_def(0,0,1,80,LNORMAL,BDR_0P);

/* Disable exiting out top of menu via ESC */

   sfm_opt(MNTOPESCAPE,OFF,menu_entry);

/* Configure the second level pop up menu */

   lev2_opt1 = mnfm_dap(MNPOP,LNORMAL,BDR_LNP);

/* Options within the top level menu */

   mnf_def(0,0,"Parameters",p_lev1_opt1,lev2_opt1,NULLFP,menu_entry);
   mnf_def(0,37,"Display",p_lev1_opt2,NULLP,adisplay,menu_entry);
   mnf_def(0,71,"Exit",p_lev1_opt3,NULLP,aexit,menu_entry);

/* Options within the second level menu */

   mnf_def(0,0,"Filenames",p_lev2_opt11,NULLP,alev2_opt11,lev2_opt1);
   mnf_def(1,0,"Select records",p_lev2_opt12,NULLP,alev2_opt12,lev2_opt1);
   mnf_def(2,0,"Display",p_lev2_opt13,NULLP,alev2_opt13,lev2_opt1);
```

```
   return;

}


===========================================================================
                                File: BATTWIND.C
===========================================================================


/*
*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*
*
*   MODULE NAME:  put_title
*
*   DESCRIPTION:  This function displays the title block prior to entering
*     the menu structure.
*
*   NOTES:
*
*   DEVELOPED BY: Troy K. Moore
*   DATE:         11/2/87
*
*   REVISION NOTES:
*
*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*
*/

#define     WN_DEBUG

#include    <io.h>
#include    <fcntl.h>
#include    <string.h>
#include    <battdefs.h>
#include    <battxtrn.h>
#include    <battxwnd.h>

void put_title()

{
   int      temp;

   char     *line1 = "    BAT Tool Radial Trace Utility\n";
   char     *line2 = "      Developed by Troy K. Moore\n\n";
   char     *line3 = "            Version: %4.2f\n";
   char     *line4 = "           Dated: %8s";

/* Prepare for writing to the title window */

   set_wn(title_wind);
   mv_cs(1,0,title_wind);
```

```
/* Place information in the window */

   v_st(line1,title_wind);
   v_st(line2,title_wind);
   v_printf(title_wind,line3,VER);
   v_printf(title_wind,line4,VER_DATE);

/* Wait for keyboard input from user before removing the window */

   temp = ki();
   unset_wn(title_wind);

   return;
}


/*
*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*
*
*   MODULE NAME:  fill_stat_wind
*
*   DESCRIPTION:  This module is designed to fill the status window with
*     default information upon algorithm start-up.
*
*   NOTES:
*
*   DEVELOPED BY: Troy K. Moore
*   DATE:         10/28/87
*
*   REVISION NOTES:
*
*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*
*/

void fill_stat_wind()

{
   int   count;

   void stat_wind_wrt();

   for (count = 1; count <= 10; count++)
      stat_wind_wrt(count);

   return;

}
```

```
/*
*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*
*
*   MODULE NAME:  stat_wind_wrt
*
*   DESCRIPTION:  This module is responsible for writing individual lines
*     of information to the status window.  Numerous other routines call
*     this function to update the status window in response to user
*     specifications.
*
*   NOTES:  The status window is composed of 10 lines, numbered 1 thru 10.
*     The passed parameter (int) should be in the range 1 - 10.
*
*   DEVELOPED BY: Troy K. Moore
*   DATE:         11/2/87
*
*   REVISION NOTES:
*
*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*
*/

void stat_wind_wrt(line_no)

int   line_no;

{
   void  mv_cs();
   char  *v_st();
   int   v_printf();

/* Update the line in the status window based on the passed value */

   switch(line_no)
   {
      case 1:                       /* Line 1 */
         mv_cs(0,0,stat_wind);
         v_st("Source",stat_wind);
         break;

      case 2:                       /* Line 2 */
         mv_cs(1,0,stat_wind);
         v_printf(stat_wind," Data Fname: %-40.40s",data_fname);
         break;

      case 3:                       /* Line 3 */
         mv_cs(2,0,stat_wind);
         v_printf(stat_wind," DP Fname:   %-40.40s",dp_fname);
         break;

      case 4:                       /* Line 4 */
         mv_cs(3,0,stat_wind);
         v_printf(stat_wind," Min depth in file: %8.2f          Max depth in file: %8.2f",
               last_file_depth,first_file_depth);
         break;
```

```
      case 5:                    /* Line 5 */
         mv_cs(4,0,stat_wind);
         v_printf(stat_wind," Start depth:   %8.21f              End depth:   %8.21f",
               init_depth,fin_depth);
         break;

      case 6:                    /* Line 6 */
         mv_cs(5,0,stat_wind);
         v_printf(stat_wind," Increment:       %6.21f              No of traces:   %d",
               inc_depth,no_traces);
         break;

      case 7:                    /* Line 7 (blank) */
         break;

      case 8:                    /* Line 8 */
         mv_cs(7,0,stat_wind);
         v_st("Display",stat_wind);
         break;

      case 9:                    /* Line 9 */
         mv_cs(8,0,stat_wind);
         v_printf(stat_wind," Total number of displays: %3d       Number traces per display:   %d",
               no_disps,traces_per_disp);
         break;

      case 10:                   /* Line 10 */
         mv_cs(9,0,stat_wind);
         v_printf(stat_wind," Base tick line:   %5.21f            Resolution between tick lines:  %5.21f",
               base_tic,tic_res);
         break;

      default:                   /* Passed argument not supported */
         break;

   }

   return;

}


/*
*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*
*
*  MODULE NAME:  BATT action functions
*
*  DESCRIPTION:  These functions are called as a direct result of items in
*     the menu being selected.  Functions then may call up a data form to
*     obtain parameters from the user.  Each function is responsible for
*     updating the status window if necessary.
*
*  NOTES:  All the action functions are initially defined in battstup.c
```

```
*
*   DEVELOPED BY: Troy K. Moore
*   DATE:         11/6/87
*
*   REVISION NOTES:
*
*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*
*/

/*>>>>>>>>>>>>>>>>>>  alev2_opt11 - Input file processing  <<<<<<<<<<<<<<<<<<<*/

int alev2_opt11(parm)

char  *parm;

{
   char        fname1[FNAME_LEN], fname2[FNAME_LEN];

   int         read_flag;

   long int    seek_flag;

   REC_INFO    temp_buf;

   void        error_handler();
   long        lseek();

/* Copy the existing file names before the form is called up */

   strcpy(fname1,data_fname);
   strcpy(fname2,dp_fname);

/* Use form to possibly obtain new file names */

   fm_proc(0,fn_form);

/* If no new names were entered, return to the menu */

   if ((strcmp(fname1,data_fname) == 0) && (strcmp(fname2,dp_fname) == 0))
      return(0);

/* Indicate that new records will have to be identified */

   rsel_flag = 1;

/* If files are open, close them */

   if (dfh != 0)
      close(dfh);
```

```
   dfh = open(data_fname,O_RDONLY|O_BINARY);
   if (dfh == -1)
   {
      error_handler(20);
      return(-1);
   }

   if (dpfh != 0)
      close(dpfh);

   dpfh = open(dp_fname,O_RDONLY|O_BINARY);
   if (dpfh == -1)
   {
      error_handler(21);
      return(-1);
   }

/* Obtain the first and last depths in the input file */

   seek_flag = lseek(dpfh,0L,SEEK_SET);
   if (seek_flag == -1L)
   {
      error_handler(22);
      return(-1);
   }

   read_flag = read(dpfh,(char *)&temp_buf,(unsigned int)sizeof(REC_INFO));
   if (read_flag < 0)
   {
      error_handler(23);
      return(-1);
   }

   first_file_depth = temp_buf.depth;

   seek_flag = lseek(dpfh,-1L*(sizeof(REC_INFO)),SEEK_END);
   if (seek_flag == -1L)
   {
      error_handler(24);
      return(-1);
   }

   read_flag = read(dpfh,(char *)&temp_buf,(unsigned int)sizeof(REC_INFO));
   if (read_flag < 0)
   {
      error_handler(25);
      return(-1);
   }

   last_file_depth = temp_buf.depth;
```

```
/* Update the information in the status window */

   stat_wind_wrt(2);
   stat_wind_wrt(3);
   stat_wind_wrt(4);

   return(0);
}

/*>>>>>>>>>>>>>>>  alev2_opt12 - Obtain source parameters  <<<<<<<<<<<<<<<<<<*/

int alev2_opt12(parm)

char  *parm;

{
   double   first, last, inc;

/* Copy the existing parameters before the form is called up */

   first = init_depth;
   last = fin_depth;
   inc = inc_depth;

/* Use form to possibly obtain new source parameters */

   fm_proc(0,rspec_form);

/* If no new parameters were entered, return to the menu */

   if ((first == init_depth) && (last == fin_depth) && (inc == inc_depth))
      return(0);

/* Indicate that new records will have to be identified */

   rsel_flag = 1;

/* Use new information to generate other parameters */

   no_traces = (int)((fin_depth - init_depth)/ inc_depth) + 1;
   fin_depth = init_depth + (inc_depth * (no_traces - 1));
   no_disps = no_traces/traces_per_disp;
   no_disps = (no_traces == (no_disps * traces_per_disp)) ?
         no_disps : no_disps + 1;

/* Update the information in the status window */

   stat_wind_wrt(5);
   stat_wind_wrt(6);
   stat_wind_wrt(9);

   return(0);
}
```

```
/*>>>>>>>>>>>>>>>  alev2_opt13 - Obtain display parameters  <<<<<<<<<<<<<<<<<<*/

int alev2_opt13(parm)

char  *parm;

{


/* Use form to obtain new display parameters */

   fm_proc(0,dspec_form);

/* Use new information to generate other parameters */

   no_disps = no_traces/traces_per_disp;
   no_disps = (no_traces == (no_disps * traces_per_disp)) ?
         no_disps : no_disps + 1;

/* Update the information in the status window */

   stat_wind_wrt(9);
   stat_wind_wrt(10);

   return(0);
}

/*>>>>>>>>>>>>>>>>>>>  adisplay - Display the traces  <<<<<<<<<<<<<<<<<<<<<<<<*/

int adisplay(parm)

char  *parm;

{
   return(-2);
}

/*>>>>>>>>>>>>>>>>>>>>>>>>>  aexit - Return to DOS  <<<<<<<<<<<<<<<<<<<<<<<<<<<<*/

int aexit(parm)

char  *parm;

{
   run_flag = -1;

   return(-2);
}
```

Appendix K - BATT Source Code

```
/*
*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*
*
*  MODULE NAME:  error_handler
*
*  DESCRIPTION:  error_handler was written to provide one central function
*    to handle the output of error messages.  This covers not only errors
*    detected by local functions but also errors returned by MSC functions.
*    This function is responsible for setting, writing an error message to,
*    and unsetting the error message window when called.
*
*  NOTES:  Passed parameter
*             index - specifies which error processing procedure to use
*                     (int)
*          Returned value - none
*
*    Error messages are allocated as follows:
*       Message number         Used by function
*           1 - 19                check_parms
*          20 - 29                alev2_opt11
*          30 - 39                sel_recs
*          40 - 49                intrvl_calc
*          50 - 59                gen_disp
*
*    Not all the message numbers allocated are actually used.  Passing a
*    non-implemented message number (index) will result in the error
*    message window being displayed with an error_handler error message.
*
*  DEVELOPED BY: Troy K. Moore
*  DATE:         11/10/87
*
*  REVISION NOTES:
*
*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*
*/
void  error_handler(index)

int   index;

{

   char   *msg1_1 = "     Initial depth selected out of bounds, must be as follows:\n";
   char   *msg1_2 = "                 %6.1f <= first depth <= %6.1f";
   char   *msg2_1 = "     Final depth selected out of bounds, must be as follows:\n";
   char   *msg2_2 = "                 %6.1f <= ending depth <= %6.1f";
   char   *msg3_1 = "     Start and end depth in wrong order, must be as follows:\n";
   char   *msg3_2 = "                   start depth <= ending depth";
   char   *msg4_1 = "        Depth increment is too small, must be as follows:\n";
   char   *msg4_2 = "                       depth increment > %2.1f";
   char   *msg5_1 = "  Too many traces per display were requested, must be as follows:\n";
   char   *msg5_2 = "                    traces per display <= %3d";
   char   *msg6_1 = "    Too many total traces were requested, must be as follows:\n";
   char   *msg6_2 = "                       total traces <= %4d";
```

```
    char *msg20_1 = "                 alev2_opt11: error opening data file\n";
    char *msg21_1 = "                 alev2_opt11: error opening dp file\n";
    char *msg22_1 = "               alev2_opt11: error during initial lseek\n";
    char *msg23_1 = "           alev2_opt11: error during initial read of dp file\n";
    char *msg24_1 = "                alev2_opt11: error during second lseek\n";
    char *msg25_1 = "           alev2_opt11: error during second read of dp file\n";

    char *msg30_1 = "         select_recs: error initial positioning file pointer\n";
    char *msg31_1 = "                 select_recs: error reading dp file\n";
    char *msg32_1 = "            select_recs: end of data reached prematurely\n";

    char *msg40_1 = "         intrvl_calc: error positioning file pointer, dp file\n";
    char *msg41_1 = "            intrvl_calc: error reading from the dp file\n";
    char *msg42_1 = "       intrvl_calc: error positioning file pointer, data file\n";
    char *msg43_1 = "           intrvl_calc: error reading from the data file\n";

    char *msg50_1 = "          gen_disp: error positioning file pointer, dp file\n";
    char *msg51_1 = "               gen_disp: error reading from the dp file\n";
    char *msg52_1 = "        gen_disp: error positioning file pointer, data file\n";
    char *msg53_1 = "              gen_disp: error reading from the data file\n";
    char *msg54_1 = "          gen_disp: error during double to string conversion\n";

    char *errno_line = "                             errno = %2d";

    char *def_line = "               error_handler: error %2d not implemented";

    int   temp;

    extern int  errno;


/* Prepare for writing to the error message window */

    set_wn(errorx_wind);
    mv_cs(1,0,errorx_wind);

/* Select which message to output based on index */

    switch (index)
    {
       case 1:
          v_st(msg1_1,errorx_wind);
          v_printf(errorx_wind,msg1_2,last_file_depth,first_file_depth);
          break;

       case 2:
          v_st(msg2_1,errorx_wind);
          v_printf(errorx_wind,msg2_2,last_file_depth,first_file_depth);
          break;
```

```
case 3:
   v_st(msg3_1,errorx_wind);
   v_st(msg3_2,errorx_wind);
   break;

case 4:
   v_st(msg4_1,errorx_wind);
   v_printf(errorx_wind,msg4_2,MIN_INC);
   break;

case 5:
   v_st(msg5_1,errorx_wind);
   v_printf(errorx_wind,msg5_2,TRACES_DISP);
   break;

case 6:
   v_st(msg6_1,errorx_wind);
   v_printf(errorx_wind,msg6_2,RECS_SEL);
   break;

case 20:
   v_st(msg20_1,errorx_wind);
   v_printf(errorx_wind,errno_line,errno);
   break;

case 21:
   v_st(msg21_1,errorx_wind);
   v_printf(errorx_wind,errno_line,errno);
   break;

case 22:
   v_st(msg22_1,errorx_wind);
   v_printf(errorx_wind,errno_line,errno);
   break;

case 23:
   v_st(msg23_1,errorx_wind);
   v_printf(errorx_wind,errno_line,errno);
   break;

case 24:
   v_st(msg24_1,errorx_wind);
   v_printf(errorx_wind,errno_line,errno);
   break;

case 25:
   v_st(msg25_1,errorx_wind);
   v_printf(errorx_wind,errno_line,errno);
   break;

case 30:
   v_st(msg30_1,errorx_wind);
   v_printf(errorx_wind,errno_line,errno);
   break;
```

```
case 31:
   v_st(msg31_1,errorx_wind);
   v_printf(errorx_wind,errno_line,errno);
   break;

case 32:
   v_st(msg32_1,errorx_wind);
   break;

case 40:
   v_st(msg40_1,errorx_wind);
   v_printf(errorx_wind,errno_line,errno);
   break;

case 41:
   v_st(msg41_1,errorx_wind);
   v_printf(errorx_wind,errno_line,errno);
   break;

case 42:
   v_st(msg42_1,errorx_wind);
   v_printf(errorx_wind,errno_line,errno);
   break;

case 43:
   v_st(msg43_1,errorx_wind);
   v_printf(errorx_wind,errno_line,errno);
   break;

case 50:
   v_st(msg50_1,errorx_wind);
   v_printf(errorx_wind,errno_line,errno);
   break;

case 51:
   v_st(msg51_1,errorx_wind);
   v_printf(errorx_wind,errno_line,errno);
   break;

case 52:
   v_st(msg52_1,errorx_wind);
   v_printf(errorx_wind,errno_line,errno);
   break;

case 53:
   v_st(msg53_1,errorx_wind);
   v_printf(errorx_wind,errno_line,errno);
   break;

case 54:
   v_st(msg54_1,errorx_wind);
```

```
      default:
         v_printf(errorx_wind,def_line,index);

   }

/* Wait for input from the keyboard before removing window and continuing */

   temp = ki();
   unset_wn(errorx_wind);

   return;

}


/*
*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*
*
*  MODULE NAME:  continue_disp
*
*  DESCRIPTION:  This module is designed to determine from the user if he
*    wishes to continue displaying additional traces or return to the menu
*    to modify parameters.
*
*  NOTES:  Passed parameters
*             count - number of current display (int)
*          Returned parameters
*             0 - continue with next display
*             1 - return to the menu
*
*  DEVELOPED BY: Troy K. Moore
*  DATE:         10/28/87
*
*  REVISION NOTES:
*
*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*
*/

int continue_disp(count)

int   count;

{
   int      temp, return_val;

/* Prepare for writing to the information window */

   set_wn(info_wind);
   mv_cs(1,0,info_wind);
```

```
/* Write display count information to the window */

   v_printf(info_wind,"   Page %d of %d total pages",count+1,no_disps);
   mv_cs(2,0,info_wind);
   v_st("   Enter Ctrl-C to return to the menu structure",info_wind);

/* Process keyboard input */

   temp = ki();
   return_val = (temp == 0x03) ? 1 : 0;

/* Remove the information window */

   unset_wn(info_wind);

/* Return to the calling routine */

   return(return_val);

}


==========================================================================
                              File: BATTPROC.C
==========================================================================


/*
*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*
*
*  MODULE NAME:  check_parms
*
*  DESCRIPTION:  This function was developed to check the parameters
*    specified by the user through the menu before the body of the
*    processing loop is executed.  If check_parms does detect an error,
*    the error handler function is called to notify the user of an
*    invalid parameter.  He/she is then sent back to the menu to modify
*    the original parameters in order to eliminate the error condition.
*
*  NOTES:  Only the first error detected is processed.
*          Returned parameter
*             0 - all parameters have passed tests
*            -1 - an invalid parameter has been detected
*
*  DEVELOPED BY: Troy K. Moore
*  DATE:         11/10/87
*
*  REVISION NOTES:
*    12/7/87  TKM   Changed error return value from 1 to -1 to be
*                   consistent with other functions.
*
*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*
*/
```

Appendix K - BATT Source Code

```
#include    <io.h>
#include    <stdio.h>
#include    <stdlib.h>
#include    <fcntl.h>
#include    <math.h>

#include    <battdefs.h>
#include    <battxtrn.h>
#include    <\image\dt_iris\isdefs.c>


int check_parms()

{
   void      error_handler();

/* Make sure that starting and ending depths are in the input data file */

   if (init_depth < last_file_depth || init_depth > first_file_depth)
   {
      error_handler(1);
      return(-1);
   }

   if (fin_depth < last_file_depth || fin_depth > first_file_depth)
   {
      error_handler(2);
      return(-1);
   }

/* Make sure starting and ending depths are in the correct order */

   if (fin_depth < init_depth)
   {
      error_handler(3);
      return(-1);
   }

/* Check the increment for valid range */

   if (inc_depth < MIN_INC)
   {
      error_handler(4);
      return(-1);
   }

/* Check the number of traces per display */

   if (traces_per_disp > TRACES_DISP)
   {
      error_handler(5);
      return(-1);
   }
```

```
/* Check the total number of traces to process */

   if (no_disps > RECS_SEL)
   {
      error_handler(6);
      return(-1);
   }

/* Return without error if this point is reached */

   return(0);

}


/*
*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*
*
*  MODULE NAME:  select_recs
*
*  DESCRIPTION:  This function is designed to select records from the data
*    file that most closely match the depth specifications given by the
*    user.  The record number of each selected record is stored in a
*    structure containing information associated with each record used.
*    Records are selected based on the absolute value of the difference
*    between the record's depth and a calculated ideal next record depth.
*
*  NOTES:  select_recs is a modified version of the function depth_rec_sel
*    used by BATD.
*    Records are numbered starting with 1.
*    Return values:    0 - no error detected
*                     -1 - error detected
*
*  DEVELOPED BY: Troy K. Moore
*  DATE:         11/9/87
*
*  REVISION NOTES:
*    12/4/87  TKM   Added code to update cur_dif and next_dif with respect
*                   to the new target depth after a depth match occurred.
*
*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*
*/

int   select_recs()

{
   int         found_count, read_bytes;

   long int    seek_flag, rec_count;

   float       cur_dif, prev_dif, next_dif, target_depth, cur_depth;

   void        error_handler();
   long int    lseek();
```

```
   #ifdef DEBUG_PRINT1
      fprintf(stdprn,"\n\r <<select_recs debug output>>\n\r");
   #endif

/* Initialize variables */

   found_count = no_traces - 1;
   target_depth = fin_depth;

/* Set the file pointer to the start of the dp file */

   seek_flag = lseek(dpfh,0L,SEEK_SET);
   if (seek_flag == -1L)
   {
      error_handler(30);
      return(-1);
   }

/* Read an entry for 'current' parameters */

   read_bytes = read(dpfh,(char *)info_buf_ptr,(unsigned int)sizeof(info_buf));
   if (read_bytes != sizeof(info_buf))
   {
      error_handler(31);
      return(-1);
   }

/* Calculate first current difference and set up first previous difference as
   loop only calculates new next difference */

   cur_dif = info_buf.depth - target_depth;
   cur_dif = (cur_dif < 0.0) ? -1.0 * cur_dif : cur_dif;

   cur_depth = info_buf.depth;

   prev_dif = 10000.0;

/* Read an entry for 'next' parameters */

   read_bytes = read(dpfh,(char *)info_buf_ptr,(unsigned int)sizeof(info_buf));
   if (read_bytes != sizeof(info_buf))
   {
      error_handler(31);
      return(-1);
   }

/* Loop until all records have been found */

   for (rec_count = 1L; found_count >= 0; rec_count++)
   {
      next_dif = info_buf.depth - target_depth;
      next_dif = (next_dif < 0.0) ? -1.0 * next_dif : next_dif;
```

```
/* Depth match */

      if (cur_dif < prev_dif && cur_dif <= next_dif && cur_dif < CHECK_BAND)
      {
         selected_recs[found_count].depth = cur_depth;
         selected_recs[found_count].rec_no = rec_count;

         target_depth = target_depth - inc_depth;  /* calc next depth */

         #ifdef DEBUG_PRINT1
            fprintf(stdprn,"found_count: %d depth: %.2f record: %ld\n\r",
                  found_count,selected_recs[found_count].depth,
                  selected_recs[found_count].rec_no);
         #endif

         found_count--;

/* Update differences as they now refer to the new target depth */

         cur_dif = cur_dif + inc_depth;
         next_dif = next_dif + inc_depth;

      }


/* Update differences in preparation for next check */

      prev_dif = cur_dif;
      cur_dif = next_dif;

      cur_depth = info_buf.depth;

/* Read next entry from the dp file */

      read_bytes = read(dpfh,(char *)info_buf_ptr,(unsigned int)sizeof(info_buf));
      if (read_bytes != sizeof(info_buf))
      {
         error_handler(31);
         return(-1);
      }
   }

/* This code reached only if all records were identified */

   return(0);

}
```

```
/*
*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*
*
*  MODULE NAME:  intrvl_calc
*
*  DESCRIPTION:  This module is designed to determine the minimum and
*    maximum caliper data values in the data records lying in the interval
*    bounded by records selected for use by select_recs.  The values
*    found are converted from caliper in mm X 10 to radius in inches and
*    stored in the selected record structure.  The min and max values in
*    a given interval are associated with the shallower bounding selected
*    record.
*
*  NOTES:  The calculation of tt_bytes assumes there are the same number
*    of amplitude and travel time data values in the record being processed.
*    This should be true for valid records, but it is hard to predict what
*    invalid records would look like.
*    Return values:    0 - no error detected
*                     -1 - error detected
*
*  DEVELOPED BY: Troy K. Moore
*  DATE:         10/28/87
*
*  REVISION NOTES:
*
*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*
*/


int intrvl_calc()

{
   int        intrvl_count, rec_count, pcount, bytes_read, wcount;

   unsigned int   min_val, max_val, tt_bytes, skip;
   unsigned int   tt_buf[POINTS_PER_TRACE];

   long           seek_flag;

   void           error_handler();
   long           lseek();


   #ifdef DEBUG_PRINT2
      fprintf(stdprn,"\n\r <<intrvl_calc debug output>>\n\r");
   #endif

/* The interval associated with the final entry is unbounded, therefore
   set the min and max values to 0 */

   selected_recs[no_traces-1].min = 0.0;
   selected_recs[no_traces-1].max = 0.0;
```

```
/* Determine the min and max values for all the remaining intervals */

   for ( intrvl_count = no_traces - 2; intrvl_count >= 0; intrvl_count--)
   {

/* Calculate how many records are in this interval */

      rec_count = selected_recs[intrvl_count].rec_no -
                  selected_recs[intrvl_count+1].rec_no - 1;

/* Position dp file file pointer at first physical record in interval */

      seek_flag = lseek(dpfh,(long)(selected_recs[intrvl_count+1].rec_no *
                  sizeof(REC_INFO)),SEEK_SET);
      if (seek_flag == -1L)
      {
         error_handler(40);
         return(-1);
      }

/* Initialize min and max values prior to beginning search */

      min_val = 65500;
      max_val = 0;

/* Search all the records in the interval for min and max values */

      for (pcount = 0; pcount < rec_count; pcount++)
      {

/* Obtain the record offset from the dp file */

         bytes_read = read(dpfh,(char *)info_buf_ptr,
               (unsigned int)sizeof(REC_INFO));
         if (bytes_read != sizeof(REC_INFO))
         {
            error_handler(41);
            return(-1);
         }

/* Calculate the number of bytes of caliper data */

         tt_bytes = (info_buf.rec_length - REV_HEAD)/2;
         skip = tt_bytes + REV_HEAD;

/* Position the data file file pointer to the start of the caliper data */

         seek_flag = lseek(dfh,(long)(info_buf.rec_offset+skip),SEEK_SET);
         if (seek_flag == -1L)
         {
            error_handler(42);
            return(-1);
         }
```

```
/* Read only the caliper data */

         bytes_read = read(dfh,(char *)tt_buf,tt_bytes);
         if (bytes_read != tt_bytes)
         {
            error_handler(43);
            return(-1);
         }

/* Scan data looking for a new min and max */

         for (wcount = 0; wcount < (tt_bytes/sizeof(int)); wcount++)
         {
            if (tt_buf[wcount] < min_val)
               min_val = tt_buf[wcount];
            if (tt_buf[wcount] > max_val)
               max_val = tt_buf[wcount];
         }
      }

/* Save the min and max values found in the this interval in the selected
   record structure */

      selected_recs[intrvl_count].min = (float)(min_val/CON_VAL);
      selected_recs[intrvl_count].max = (float)(max_val/CON_VAL);

      #ifdef DEBUG_PRINT2
         fprintf(stdprn,"interval from rec %ld to rec %ld (%d recs)\n\r",
               selected_recs[intrvl_count+1].rec_no,
               selected_recs[intrvl_count].rec_no,rec_count);
         fprintf(stdprn,"min: %x max: %x\n\r",min_val,max_val);
      #endif

   }

   return(0);

}

/*
*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*
*
*  MODULE NAME:  calc_tick_lines
*
*  DESCRIPTION:  This module is designed to take the display parameters
*    and determine where in the display area the tick lines should
*    be placed.
*
*  NOTES:
*
*  DEVELOPED BY: Troy K. Moore
*  DATE:         10/28/87
*
```

```
*   REVISION NOTES:
*
*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*
*/

void calc_tick_lines()

{
   int      dlines_per_band, start_tick_line, count;


   #ifdef DEBUG_PRINT3
      fprintf(stdprn,"\n\r <<calc_tick_lines debug output>>\n\r");
   #endif

/* Calculate the layout parameters */

   dlines_per_band = DISP_LINES/(traces_per_disp + 1);
   start_tick_line = ((DISP_LINES % (traces_per_disp + 1))/2) - 1;

   if (start_tick_line == -1)
   {
      dlines_per_band--;
      start_tick_line = ((DISP_LINES-(dlines_per_band*(traces_per_disp+1)))/2)-1;
   }

   dlines_per_inch = dlines_per_band/tic_res;

   #ifdef DEBUG_PRINT3
      fprintf(stdprn,"dlines_per_band: %d start_tick_line: %d dlines_per_inch: %.2f\n\n\r",
            dlines_per_band,start_tick_line,dlines_per_inch);
   #endif

/* Fill in row values for tick line end points */

   for (count = 0; count < (traces_per_disp + 2); count++)
      tick_lines[count][0] = start_tick_line + (count * dlines_per_band);

   return;

}


/*
*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*
*
*   MODULE NAME:  gen_disp
*
*   DESCRIPTION:  This module is designed to obtain and map the selected
*     record caliper data into the display coordinate system.  Each trace
*     is mapped with respect to a primary tick line.  As each trace is
*     mapped, it is drawn on the display.  After all traces have been
*     processed, all the tick lines are drawn.  The final task is to annotate
*     each trace.
```

```
*
*  NOTES:
*    Return values:    0 - no error detected
*                     -1 - error detected
*
*  DEVELOPED BY: Troy K. Moore
*  DATE:         11/9/87
*
*  REVISION NOTES:
*
*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*
*/

int gen_disp(loop_count)

int      loop_count;

{

   int      trace_limit, trace_count, srec_index, bytes_read, pixel_count;
   int      start_col, start_row, draw_count, disp_limit, ret_flag, tic_offset;
   int      stra_count, strb_count, strc_count, strd_count;
   int      stra[15], strb[15], strc[15], strd[15];

   static int  depth_str[] = {'D','e','p','t','h'};
   static int  ref_str[]   = {'R','e','f'};
   static int  min_str[]   = {'M','i','n'};
   static int  max_str[]   = {'M','a','x'};

   unsigned int   tt_bytes, skip;
   unsigned int   tt_buf[POINTS_PER_TRACE];

   long     seek_flag;

   double   inch_rad;

   void     error_handler();
   int      doub_convrt();
   long     lseek();


   #ifdef DEBUG_PRINT4
      fprintf(stdprn,"\n\r <<gen_disp debug output>>\n\r");
   #endif

/* Select intensity for drawing the traces */

   is_set_foreground(TRACE_INTENSITY);

/* Calculate how many traces to display */

   trace_limit = (loop_count + 1 < no_disps) ? traces_per_disp :
                  no_traces - (loop_count * traces_per_disp);
```

```
/* Calculate the maximum row value that data can be assigned */

   disp_limit = DISP_LINES - 1;

/* Generate a pointer into the selected record structure */

   srec_index = loop_count * traces_per_disp;

/* Generate and display a display full of traces */

   for (trace_count = 0; trace_count < trace_limit; trace_count++)
   {

/* Using the record number in the selected record structure, get the offset
   of the selected record */

      seek_flag = lseek(dpfh,(long)((selected_recs[srec_index].rec_no - 1) *
                  sizeof(REC_INFO)),SEEK_SET);
      if (seek_flag == -1L)
      {
         error_handler(50);
         return(-1);
      }

      bytes_read = read(dpfh,(char *)info_buf_ptr,(unsigned int)sizeof(REC_INFO));
      if (bytes_read != sizeof(REC_INFO))
      {
         error_handler(51);
         return(-1);
      }

/* Calculate then read the number of bytes from the selected record that
   represents only the caliper data */

      tt_bytes = (info_buf.rec_length - REV_HEAD)/2;
      skip = tt_bytes + REV_HEAD;

      #ifdef DEBUG_PRINT4
         fprintf(stdprn,"index: %d record to map: %ld read offset: %lx read bytes: %u\n\r",
               srec_index,selected_recs[srec_index].rec_no,
               (info_buf.rec_offset+skip),tt_bytes);
      #endif

      seek_flag = lseek(dfh,(long)(info_buf.rec_offset+skip),SEEK_SET);
      if (seek_flag == -1L)
      {
         error_handler(52);
         return(-1);
      }
```

```
      bytes_read = read(dfh,(char *)tt_buf,tt_bytes);
      if (bytes_read != tt_bytes)
      {
         error_handler(53);
         return(-1);
      }

/* Map all the caliper values read into row values */

      for (pixel_count = 0; pixel_count < (tt_bytes/sizeof(int)); pixel_count++)
      {
         inch_rad = (double)(tt_buf[pixel_count])/(double)(CON_VAL);
         tic_offset = (int)((inch_rad-base_tic)*(double)dlines_per_inch);
         map_data[pixel_count][0] = tick_lines[trace_count+1][0] - tic_offset;

/* Make sure the mapped values lie within the display area */

         if (map_data[pixel_count][0] < 0)
            map_data[pixel_count][0] = 0;

         if (map_data[pixel_count][0] > disp_limit)
            map_data[pixel_count][0] = disp_limit;

         #ifdef DEBUG_PRINT4
            if (pixel_count == 0)
            {
               fprintf(stdprn,"trace count: %d raw data value: %x\n\r",
                     trace_count,tt_buf[pixel_count]);
               fprintf(stdprn,"radi in inches: %.2f offset from tic line: %d\n\r",
                     inch_rad,tic_offset);
               fprintf(stdprn,"tic line ref: %d mapped value: %d\n\n\r",
                     tick_lines[trace_count+1][0],map_data[pixel_count][0]);
            }
         #endif

      }

/* Draw the trace on the display */

      start_row = map_data[0][0];
      start_col = map_data[0][1];
      draw_count = (tt_bytes/sizeof(int)) - 1;

      is_set_graphic_position(start_row,start_col);
      is_draw_lines(DISP_FRAME,draw_count,&map_data[1][0]);

/* Update index into the selected record structure */

      srec_index++;
   }
```

```
/* Draw all the tick lines */

   is_set_foreground(TICK_LINE_INTENSITY);
   draw_count = 1;

   for (trace_count = 0; trace_count < (trace_limit + 2); trace_count++)
   {
      is_set_graphic_position(tick_lines[trace_count][0],START_COLUMN);
      is_draw_lines(DISP_FRAME,draw_count,&tick_lines[trace_count][0]);
   }

/* Calculate offset to locate the min and max values */

   start_row = (tick_lines[1][0] - tick_lines[0][0]) / 2;

/* Re-calculate the index into the selected record structure */

   srec_index = loop_count * traces_per_disp;

/* Set the intensity in which the text will be written */

   is_set_foreground(TEXT_INTENSITY);

/* Write column label information */

   is_set_graphic_position(0,DEPTH_COL);
   is_draw_text(DISP_FRAME,sizeof(depth_str)/sizeof(int),depth_str);

   is_set_graphic_position(0,REF_COL);
   is_draw_text(DISP_FRAME,sizeof(ref_str)/sizeof(int),ref_str);

   is_set_graphic_position(0,MIN_COL);
   is_draw_text(DISP_FRAME,sizeof(min_str)/sizeof(int),min_str);

   is_set_graphic_position(0,MAX_COL);
   is_draw_text(DISP_FRAME,sizeof(max_str)/sizeof(int),max_str);

/* Convert the tick line base value as it will not change between traces */

   ret_flag = doub_convrt(base_tic,1,strb,&strb_count);
   if (ret_flag != 0)
   {
      error_handler(54);
      return(-1);
   }

/* Annotate all the traces */

   if (loop_count == 0)               /* first display page */
   {
```

```
    for (trace_count = 0; trace_count < trace_limit; trace_count++)
    {
       ret_flag = doub_convrt((double)(selected_recs[srec_index].depth),2,
             stra,&stra_count);
       if (ret_flag != 0)
       {
          error_handler(54);
          return(-1);
       }

       ret_flag = doub_convrt((double)(selected_recs[srec_index].min),2,
             strc,&strc_count);
       if (ret_flag != 0)
       {
          error_handler(54);
          return(-1);
       }

       ret_flag = doub_convrt((double)(selected_recs[srec_index].max),2,
             strd,&strd_count);
       if (ret_flag != 0)
       {
          error_handler(54);
          return(-1);
       }

       is_set_graphic_position(tick_lines[trace_count+1][0]-HALF_CHAR,
             DEPTH_COL);
       is_draw_text(DISP_FRAME,stra_count,stra);
       is_set_graphic_position(tick_lines[trace_count+1][0]-HALF_CHAR,
             REF_COL);
       is_draw_text(DISP_FRAME,strb_count,strb);
       is_set_graphic_position((tick_lines[trace_count+1][0]+start_row-
             HALF_CHAR),MIN_COL);
       is_draw_text(DISP_FRAME,strc_count,strc);
       is_set_graphic_position((tick_lines[trace_count+1][0]+start_row-
             HALF_CHAR),MAX_COL);
       is_draw_text(DISP_FRAME,strd_count,strd);

       srec_index++;
    }
 }

 else                                   /* all other display pages */
 {
    for (trace_count = 0; trace_count < trace_limit; trace_count++)
    {
       ret_flag = doub_convrt((double)(selected_recs[srec_index].depth),2,
             stra,&stra_count);
       if (ret_flag != 0)
       {
          error_handler(54);
          return(-1);
       }
```

```
      ret_flag = doub_convrt((double)(selected_recs[srec_index-1].min),2,
            strc,&strc_count);
      if (ret_flag != 0)
      {
         error_handler(54);
         return(-1);
      }

      ret_flag = doub_convrt((double)(selected_recs[srec_index-1].max),2,
            strd,&strd_count);
      if (ret_flag != 0)
      {
         error_handler(54);
         return(-1);
      }

      is_set_graphic_position(tick_lines[trace_count+1][0]-HALF_CHAR,
            DEPTH_COL);
      is_draw_text(DISP_FRAME,stra_count,stra);
      is_set_graphic_position(tick_lines[trace_count+1][0]-HALF_CHAR,
            REF_COL);
      is_draw_text(DISP_FRAME,strb_count,strb);
      is_set_graphic_position((tick_lines[trace_count+1][0]-start_row-
            HALF_CHAR),MIN_COL);
      is_draw_text(DISP_FRAME,strc_count,strc);
      is_set_graphic_position((tick_lines[trace_count+1][0]-start_row-
            HALF_CHAR),MAX_COL);
      is_draw_text(DISP_FRAME,strd_count,strd);

      srec_index++;
   }

   if (loop_count != (no_disps - 1))   /* all full display pages have */
                                       /* one extra min/max set */
   {
      ret_flag = doub_convrt((double)(selected_recs[srec_index-1].min),2,
            strc,&strc_count);
      if (ret_flag != 0)
      {
         error_handler(54);
         return(-1);
      }

      ret_flag = doub_convrt((double)(selected_recs[srec_index-1].max),2,
            strd,&strd_count);
      if (ret_flag != 0)
      {
         error_handler(54);
         return(-1);
      }

      is_set_graphic_position((tick_lines[trace_count+1][0]-start_row-
            HALF_CHAR),MIN_COL);
```

```
        is_draw_text(DISP_FRAME,strc_count,strc);
        is_set_graphic_position((tick_lines[trace_count+1][0]-start_row-
              HALF_CHAR),MAX_COL);
        is_draw_text(DISP_FRAME,strd_count,strd);
      }
   }

/* Return to calling routine */

   return(0);

}

/*
*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*
*
*   MODULE NAME:  doub_convrt
*
*   DESCRIPTION:  This function is designed to convert a value of type
*     double into an integer array of ASCII values.  This conversion is
*     required in preparation for calls to the DT-IRIS subroutine
*     is_text_draw which uses an integer array as opposed to a character
*     array.
*
*   NOTES:  Passed parameters:
*              dval    - value to be converted (double)
*              dplaces - number of decimal places to the right of the
*                          decimal point to convert (int)
*           Returned parameters:
*              svals - integer array to be filled with ASCII values
*                        (pointer)
*              count - number of valid characters placed in array pointed
*                        to by svals (pointer)
*           Error flag:
*              0 - no error detected
*             -1 - dval out of range
*             -2 - dplaces out of range
*
*     The following limits are hardcoded
*              999 999 999.0 > dval > -999 999 999.0
*              0 <= dplaces <= 4
*
*     Due to these limits, the array pointed to by svals must be declared
*     15 integer units in length in the calling procedure.
*
*   DEVELOPED BY: Troy K. Moore
*   DATE:         11/6/87
*
*   REVISION NOTES:
*
*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*
*/
```

Appendix K - BATT Source Code

```
int doub_convrt(dval,dplaces,svals,count)

int      dplaces, *count, *svals;

double   dval;

{
   int      char_count, zero_flag, check_count, check_limit, quotient;

   double   divisor;

   double   fabs(), pow();

/* Make sure that passed values are within specified range */

   if (dval > 999999999.0 || dval < -999999999.0)
      return(-1);

   if (dplaces > 4 || dplaces < 0)
      return(-2);

/* Initialize parameters */

   char_count = 0;
   zero_flag = 0;
   check_limit = 0;

/* Handle any negative values */

   if (dval < 0.0)
   {
      *svals = '-';
      svals++;
      char_count++;
      dval = fabs(dval);
   }

/* Convert all digits to the left of the decimal point */

   for (check_count = 8; check_count >= check_limit; check_count--)
   {
      divisor = pow((double)10.0,(double)check_count);
      quotient = (int)(dval/divisor);

      if (quotient > 0)
      {
         zero_flag = 1;
         dval = dval - (quotient * divisor);
      }
```

```
        if (zero_flag == 1)
        {
           *svals = quotient + 0x030;
           svals++;
           char_count++;
        }
    }

/* Add the decimal point */

    *svals = '.';
    svals++;
    char_count++;

/* Convert all digits to the right of the decimal point */

    check_limit = dplaces;

    for (check_count = 1; check_count <= check_limit; check_count++)
    {
       divisor = pow((double)10.0,(double)(check_count * -1));
       quotient = (int)(dval/divisor);

       if (quotient > 0)
          dval = dval - (quotient * divisor);

       *svals = quotient + 0x030;
       svals++;
       char_count++;
    }

/* Return to the calling routine */

    *count = char_count;

    return(0);

}
```