ERD DATA-PROCESSING SOFTWARE

Reference Manual *Version 2.00*

Michael Sayers

June 1987

Engineering Research Division

UMTRI The University of Michigan Transportation Research Institute

Technical Report Documentation Page

1. Report No.	2. Government Accession No.	3. Recipient's Catalog No.
UMTRI-87-2		

4. Title and Subtitle	5. Report Date
ERD DATA-PROCESSING SOFTWARE REFERENCE MANUAL VERSION 2.00	June 1987
	6. Performing Organization Code
7. Author(s)	8. Performing Organization Report No.
Michael Sayers	UMTRI-87-2
9. Performing Organization Name and Address	10. Work Unit No. (TRAIS)
The University of Michigan Transportation Research Institute 2901 Baxter Road, Ann Arbor, Michigan 48109	11. Contract or Grant No.
	13. Type of Report and Period Covered
12. Sponsoring Agency Name and Address	Version 2.00 9/84 - 6/87
	14. Sponsoring Agency Code

15. Supplementary Notes

16. Abstract

This manual describes computer software and specifications that have been developed within the Engineering Research Division (ERD) at The University of Michigan Transportation Research Institute (UMTRI) for the purpose of processing data in computer files for applications where the data are organized by channel number and sample number. The software mainly involves graphics routines that facilitate viewing the data. Other software deals with accessing the data files, and performing transforms on the data such as spectral analysis.

A standard ERD file format is described that is appropriate for this type of data. Data-processing programs for ERD files obtain most of the information needed from the file itself. As a result, a program that can deal with one ERD file can usually deal with all ERD files, regardless of their origin or application.

The programs described in this manual are written in FORTRAN 77 and are available on The University of Michigan mainframe computer system (MTS). They include libraries with over 80 subroutines and functions, along with various stand-alone programs.

17. Key Words	18. Distribution Statement
Data Processing, Standard Computer Files, Plotting Software, Signal Processing, Graphics, Spectral Analysis, Vehicle Simulation	UNLIMITED

19. Security Classif. (of this report)	20. Security Classif. (of this page)	21. No. of Pages	22. Price
NONE	NONE	109	

TABLE OF CONTENTS

LIST OF FIGURES

Figure *Page*

LIST OF TABLES

FOREWORD

Over the years, a great deal of computerized data processing has been performed in various research projects by the Engineering Research Division (ERD) at The University of Michigan Transportation Research Institute (UMTRI). In some, computer simulations are used to predict vehicle behavior. In others, experimental measurements from the laboratory and test track are made with computer-based data-acquistion systems. For these data to be understandable, they must usually be viewed by engineers in graphical form. Although the variables of interest are unique for each project, the process of preparing plots from a data file is nearly the same in every case, regardless of the source of the data.

In 1984, several major research projects began which required extensive data processing. At that time, a generalized file format was designed for data consisting of sampled values of one or more variables. For lack of a better name, this is called the ERD file format. In addition to the data and layout of the data, ERD files also include a great deal of labelling information to facilitate the automatic generation of engineering plots. They are essentially self-documenting, such that data-processing programs can obtain all pertinent information from the file. Thus, a data-processing program that can deal with one ERD file can usually deal with all ERD files, regardless of their size or source.

Once the standard file format was in use, an existing graphics package was modified to simplify the preparation of engineering plots of data from these files. As specific bells and whistles were needed for different research projects, they were added to the plotter. The ERD plotter has become very easy to use to interactively view data on line, and also to process large batches of data to obtain hard copies for later viewing. Generally, the plots produced are in "final form" so that they can be used directly in reports and published papers.

This manual was written for two purposes: (1) as documentation for the ERD software as it now exists on The University of Michigan mainframe computer system, and (2) as a description of the ERD file format and some of the existing software that takes advantage of that format. The problem of viewing and manipulating measured and computed variables is one that reoccurs in many fields outside of that of vehicle dynamics. The ERD file format and the associated software are a solution that might prove helpful to other engineers confronting the similar needs.

1. INTRODUCTION

To understand complex phenomena measured in the laboratory or simulated by computer, they must usually be viewed by engineers in graphical form. A reoccurring problem has been that various software packages have different requirements regarding the format of the data files. Further, data produced from different sources are typically stored using formats specific to each source.

This manual describes a one-size-fits-all file format that has been developed within the Engineering Research Division (ERD) at The University of Michigan Transportation Research Institute (UMTRI). It also describes data-processing software that has general utility in the fields of vehicle dynamics and measuring road roughness.

The so-called ERD file format is appropriate for disk or tape files whenever data are organized by channel number and sample number. The data in the file are stored in the same form as would be used for a multi-track tape recorder. The file also contains the information that would normally be put in a log sheet summarizing the data, and can optionally include labeling information needed for preparing graphical plots of the data. Data processing programs for ERD files obtain most of the information needed from the file itself. As a result, software that can deal with one ERD file can usually deal with all ERD files, regardless of their source or application.

A substantial amount of software has been written to simplify the viewing and manipulation of data in the ERD files. By far, the main advantage of this format has been the ease of use with which data can be viewed. A plotter has been developed and improved to the extent that it is remarkably simple to view data interactively, or to obtain hard copies of hundreds of plots using a batch mode. Because the files contain the necessary labeling information, the plotter requires minimal information from the user. Typically, an engineer will specify only the name of a file and the channel(s) of interest to view.

Most of the applications involving ERD files have been performed using The University of Michigan mainframe (terminal) computer system (MTS). For the time being, most of the data processing activities take place on MTS.[1] However, the ERD file format is not specific to MTS, and the format is also appropriate for other computers. ERD files are presently created by some of the UMTRI laboratory equipment based on the IBM PC microcomputer, and most of the software from MTS is being ported to the Apple Macintosh and the IBM PC. All of the software described in this manual is written in FORTRAN 77, and should run on most computers with little modification. (The major modification is to find equivalent primitive graphics routines to perform functions such as drawing a line, writing text, moving a pen, etc.)

The next section (Section 2) in this manual is the reference for the ERD file layout, which is of interest to persons writing computer programs that read or write ERD files. The remainder of the manual describes several major programs and over 80 library functions and subroutines that are useful when dealing with data-processing applications

[1] In order to use the ERD software on MTS, some familiarity with MTS is assumed. The necessary details concerning the use of MTS are provided in the publication "Introduction to MTS," 1984, available from The University of Michigan Computing Center.

and/or ERD files. Section 3 describes the plotting program which runs on MTS, including example sessions and example plots. Section 4 describes how to read and write ERD files using programs written in FORTRAN. It also describes a toolbox of FORTRAN subroutines that can reduce the programming required to develope data processing software. Section 5 describes several utility programs for handling ERD files on MTS, such as copying, splitting, and converting. The next sections describe software developed for data-processing applications in general. Section 6 describes a number of plotting subroutines that perform functions such as drawing axes and determining scale factors. It also describes two plotting programs that were written before ERD files were developed, and which can be used to plot data in arbitrary computer files or as defined by mathematical functions. Section 7 describes a library of subroutines used for improving the user interface of the FORTRAN 77 programs, by offering capabilities similar to those in "friendlier" languages such as BASIC or PASCAL. Section 8 describes software written to process signals, include simple band-pass filtering, quarter-car simulation, and Fourier analysis. Section 9 describes the outputs from the major vehicle simulation programs used within ERD—the *Yaw-Roll* and *Phase 4* models. It also describes the convention used in ERD programs for labeling channels applicable for heavy trucks.

2. ERD FILES

The Engineering Research Division (ERD) at The University of Michigan Transportation Research Institute (UMTRI) has developed a standard file format to simplify the processing of data from varied sources, such as experiments, simulations, and data-processing programs. These are presently called ERD files. The file contains two independent sections, the header and data, as illustrated in Figure 1. Depending on the design of the computer operating system, the two sections may reside in the same file or in two separate files. On MTS, the mainframe computer system at The University of Michigan, both sections are always included in the same file. On the IBM PC and Apple Macintosh, it is sometimes more convenient to have two separate files having the same base name but with different extensions.

The data section contains nothing but numbers and is organized in a form similar to a multitrack tape recorder. The data are stored in binary form when efficiency is important. Alternatively, the data can be stored in text form, to facilitate transporting the files between different computers. In Figure 1, the data are shown organized in columns and rows, where the columns correspond to separate channels and the rows correspond to samples taken of all of the channels at a particular instant. The organization shown is merely a convenience for the illustration—the actual layout is a stream of numbers beginning with the first sample of channel 1, followed by the first sample of channel 2, and ending with the last sample of the last channel.

The header section of the file contains the information needed to read the numerical data. This design allows a data-processing program to first read the header information that maps out the file, and then read the rest of the file. Thus, a program that can deal with one ERD file can usually deal with any ERD file, even though the other files have different numbers of channels and perhaps different amounts of information included in the headers.

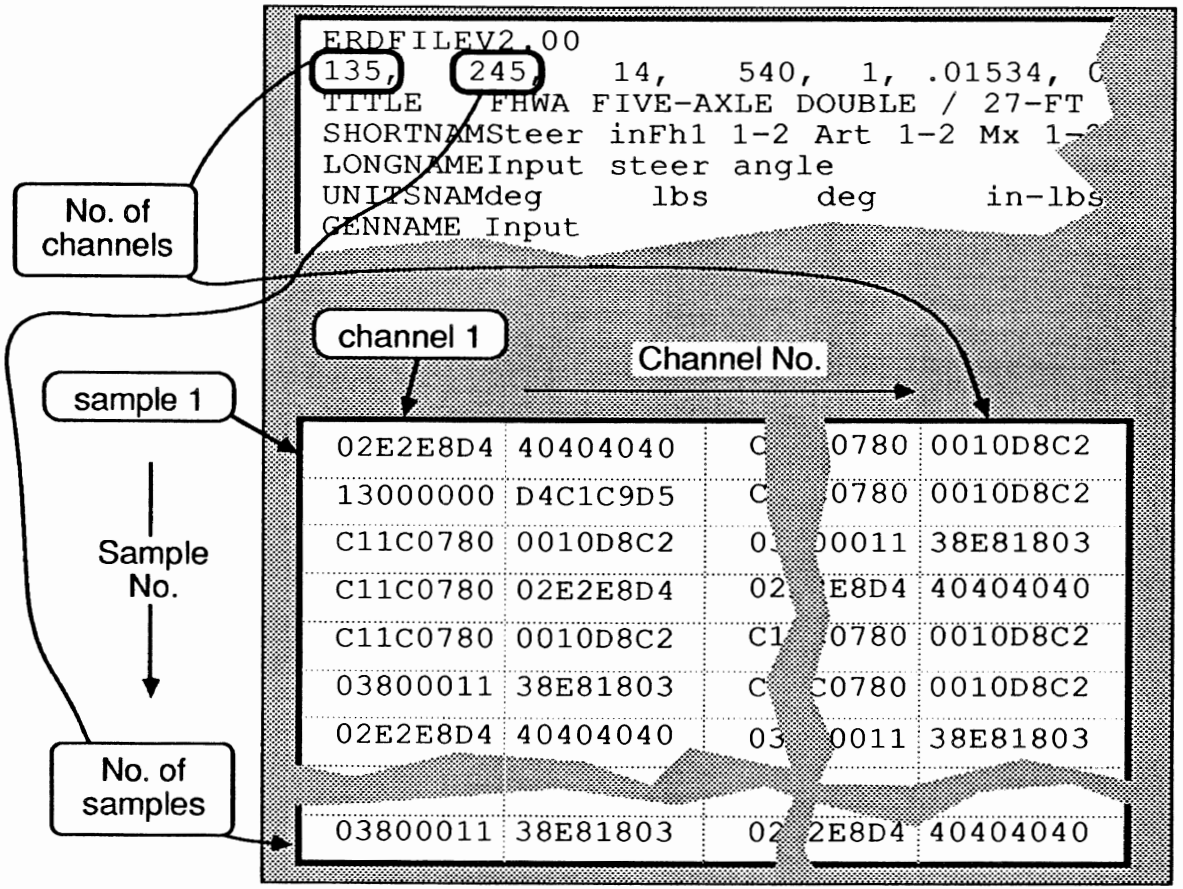
When ERD files are stored on tape, it has proven convenient to have a directory of the files on disk, so that the tape can be positioned more rapidly. It also allows one to modify the header of an ERD file using a normal text editor. For example, to change the units of a channel from feet to meters, the scale factor and the name of the units in the header can be changed while leaving the data portion of the file intact.

2.1 The Header

The header part of an ERD file consists of a series of conventional text lines that are "human readable" using conventional text editors.[1]

---

[1] On most computers text files are stored using the ASCII representation of text characters. On IBM mainframe computers, text files are stored using the EBCDIC convention. The header is stored using the representation appropriate for the computer. In this manual, ASCII and EBCDIC files are both referred to as "text."



Header

The beginning of the file describes the data, specifying: size, layout, names of channels, scale factors, and additional information

Numerical Data

The second part of the file contains numerical data, in a layout similar to a multi-channel tape recorder. Normally, the data are stored as binary numbers for maximum efficiency.

Figure 1. Example view of pieces of an ERD file.

As a minimum, the header contains three lines of text. The first line identifies the file as following the ERD format.[2] The second line describes the way that the numerical data are stored in the data section of the file. The third required line is an END statement that indicates the end of the header portion. Table 1 summarizes the lines in an ERD file, and describes the parameters used in line #2 to describe the numerical data. (These parameters are described in Subsection 2.2). Additional lines can optionally be included to provide information specific to various applications. The information in the optional records is always identified by an eight-character keyword that begins the line. If the computer program reading the file has a use for that information, then it will recognize the keyword. If not, the line is skipped. Tables 2 and 3 show two examples of headers. The header in Table 2 is fairly brief, containing only two optional lines. One provides a title for the file, and the other provides short names of eight characters each for the two channels in the file. Table 3 shows a more typical header with additional optional lines.

The header contains names and numerical values of parameters. Each name has a preassigned length, in terms of the number of characters it contains. For example, the title is defined as having 80 characters. Usually, the name will be shorter than the space allowed. When several names are on the same line, the names should be padded with blanks as needed so that following names begin at the correct column positions. For example, the header shown in Table 3 includes unit names, as identified with the keyword UNITSNAM. The name of units for the first channel, deg, has only three characters. Thus, it is followed by five spaces so that the name for the second channel, g's, begins in the correct column position.

There is no parsing performed when reading names from the header. Thus, names can include spaces, commas, and "special" characters. For example, the long names shown in the example of Table 3 include commas and spaces.

When a keyword is associated with a list of numbers, the numbers must be separated with commas. The space allotted to each number (including the comma and any spaces) can be up to 20 characters. Decimal points are optional for floating point numbers, but must never be used for integers.

To date, all of the keywords that have been used involve five data types: integers, floating point (real) numbers, 8-character names, 32-character names, and 80-character names. The number of data items is either one per file, one per channel, an arbitrary number N, or repeatable.

Table 4 lists all of the keywords that are presently in use, and Table 3 shows examples of many types of keywords. If a file contains lines beginning with 8-character combinarions that are not shown in Table 4, the contents of those lines are not recognized by existing data-processing programs.

---

[2] This manual describes Version 2.00 of the ERD file. Versions 2.00 and 1.00 contain the same information, but the Version 2.00 files are somewhat easier to prepare and read. Version 1.00 is described in the Appendix. Software on MTS will read either version automatically.

Table 1. Summary of Records in an ERD File Header.

Line No.	*Description*
1	*ERDFILEV2.00* — identifies file as having ERD format
2	NCHAN, NSAMP, NRECS, NBYTES, KEYNUM, STEP, KEYOPT — use commas to separate numbers NCHAN [integer] = Number of data channels NSAMP [integer] = Number of samples for each channel. The total number of sampled values in the data portion of the file is NCHAN × NSAMP. (If unknown, use –1.) NRECS [integer] = Number of records of data. (record ≈ line) Ignored for text data (KEYNUM = 5.) (If unknown, use –1.) NBYTES [integer] *binary data:* Number of bytes per record. Should be chosen such that each record begins with channel 1: that is, NBYTES = K × NCHAN × B, where K is an integer and B is the number of bytes/number (B=2 for integer, B=4 for floating-point). *text data:* Number of samples per record. Thus each record contains NBYTES × NCHAN numbers. KEYNUM[integer] = Indicates how the data are stored. 0 = 2-byte integer (binary), 1 = 4-byte floating point (binary), 5 = Formatted floating-point (text) The format must be specified using the FORMAT keyword. STEP [real] = sample interval (e.g., time step) KEYOPT [integer] = number used in some data processing applications.
•	**Optional records**. Each record begins with an 8-character keyword, followed by information associated with that keyword. Table 4 lists keywords that have been used to date.
NHEAD	*END* — indicates the end of the header
1+ NHEAD	**First** data record.
NHEAD + NRECS	**Last** data record.

Table 2. Example of a Short Header for an ERD File with Binary Data.

```
ERDFILEV2.00
      2,    501,      1,   4008,      1, 2.000000E-02,      0,
TITLE   Tanker making a J-Turn and rolling over.
SHORTNAMRoll #2 Ay cg #2
END
```

Table 3. Example of a Typical Header for an ERD File with Text Data.

```
ERDFILEV2.00
      2,    501,    167,      3,      5, 2.000000E-02,      0,
TITLE   Tanker, from simulation, rolling over.
SHORTNAMRoll #2 Ay cg #2
LONGNAMERoll Angle, Semi-trailer        Lat. Accel., Semi-trailer
UNITSNAMdeg     g's
GENNAME Roll                            Lateral Acceleration
XLABEL  time
XUNITS  sec
FORMAT  (3(2G13.6))
AXLETRAK      5,  80.0000    ,  71.5000    ,  71.5000    ,  71.5000    ,  71.5000    ,
FSTAXLES      1,      4,
HISTORY Data generated with Phase 4 model on SAT MAR 14/87   12:50:32
HISTORY There can be multiple HISTORY lines in a header.
NAXLES        5,
RIGIBODYSemi-trailer                    Semi-trailer
SPEEDMPH  35.0000    ,
END
  .000000      .000000      .000000      .000000     0.100000E-02  .000000
 0.100000E-02  .000000     0.100000E-02  .000000     0.200000E-02  .000000
 0.200000E-02  .000000     0.200000E-02  .000000     0.200000E-02  .000000
```

Table 4. Summary of Existing Keywords.

Keyword Name	*Description*	*No. of Values*	*Variable Type*
VERSION	Line 1 in header file.	1	char*32
LINE 2	Line 2 in header file. Item #6 (step) is real, all other items are integer	7	integer,real
	(The following 4 lines are strongly recommended for inclusion in all ERD files)		
TITLE	Title used for file.	1	char*80
SHORTNAM	Short names for channels.	NCHAN	char*8
LONGNAME	Long names for channels.	NCHAN	char*32
UNITSNAM	Names for units used for channels.	NCHAN	char*8
AXLETRAK	Dimensions for each axle, in inches.	NLIST	real
AXLEWT	Weights of each axle assembly, in pounds.	NLIST	real
FSTAXLES	The number of the first axle on the units in a multiple-vehicle combination.	NLIST	integer
FORMAT	FORTRAN FORMAT statement when file contains text data. Ex: (4F10.4)	1	char*32
GAIN	Gains for channels. (Default = 1.) Usually required for integer*2 data.	NCHAN	real
GENNAME	Generic names for variables, used for labelling Y axis when several variables are plotted on the same axis (ex: Position).	NCHAN	char*32
HISTORY	Additional information about the history of the ERD file.	Repeats	char*80
HITCHKEY	Codes that give the hitch types for a vehicle simulation.	NLIST	integer
NAXLES	Total number of axles in a vehicle.	1	integer
OFFSET	Offsets for channels. (default = 0.) Usually required for integer*2 data.	NCHAN	real

NOTES: Keywords must be capitilized. Spaces are significant, including leading spaces.

Numbers in a list must be separated by commas. Each number (digits, spaces, decimal point, comma) can occupy up to 20 characters, inclusive. Decimal points must not be used for integers.

Table 4. Summary of Existing Keywords — continued.

Keyword Name	*Description*	*No. of Values*	*Variable Type*
PROFINST	Name of profilometer instrument used to measure data	1	char*32
ROLLCNTR	Heights of roll centers above ground, in inches.	NLIST	real
ROLLHT	Vertical distances between the roll centers of each axle and the c.g. of the corresponding sprung mass. The units are inches.	NLIST	real
RIGIBODY	Name of rigid body associated with each variable (e.g., Axle 2, Tractor, ...)	NCHAN	char*32
SPEEDMPH	Speed associated with data, in mile/hr.	1	real
SPRUNGWT	Weights of sprung masses, in pounds.	NLIST	real
STEERIN	Name of steering input to vehicle (e.g., trap steer).	1	char*32
TESTID	Number used to identify a test.	1	real
TRUCKSIM	Name of the simulation model.	1	char*32
WHOBLAME	Person to contact with questions about the data.	1	char*32
XLABEL	Name of independent variable in ERD file (e.g., time).	1	char*32
XSTART	Starting value of independent variable. At each sample i, the X value is: X = (i-1) * STEP + XSTART	1	real
XUNITS	Units of independent variable (e.g., sec).	1	char*8

NOTES: Keywords must be capitilized. Spaces are significant, including leading spaces.

Numbers in a list must be separated by commas. Each number (digits, spaces, decimal point, comma) can occupy up to 20 characters, inclusive. Decimal points must not be used for integers.

2.2 The Data Section

The data part of the ERD file contains nothing but numbers, organized into columns and rows. The sequence of storage is:

$$X_1(1), X_2(1), \ldots X_{NCHAN}(1), X_1(2), X_2(2) \ldots X_{NCHAN}(NSAMP)$$

where NCHAN is the number of channels and NSAMP is the number of samples. The total number of sampled data values is thus NCHAN × NSAMP. All of the numbers in the data portion are stored in the same format, and there can be no "missing values."

Reading and writing binary data is very efficient, because the computer does not need to perform any conversions or transformations as the data values are moved between the file and the computer memory. When a binary format is used, the data portion of an ERD file is a direct copy of a portion of the computer memory, corresponding to a two-dimensional array having dimensions corresponding to the number of channels and the number of samples. As indicated in Table 1, two forms of binary data are presently supported on MTS: integer*2 and real*4. Integer*2 data are typically obtained by data-acquisition systems. Each integer value is a sampled reading obtained from a digitizer during a test. For most engineering applications, data are stored (in the computer memory) in real*4 format, also known as single-precision floating point. The real*4 format is commonly used for any processed data, or for data generated by the computer. The maximum efficiency for data processing is usually obtained when the real*4 format is used.

When the data are stored in binary form, the total number of bytes in the data portion of the file is NCHAN × NSAMP × (BYTES/SAMPLE). For real*4 data, there are four bytes/sample; for integer*2 data, there are two bytes/sample. On some computer systems, a file can contain a large amount of continuous binary data. In tape files, however, there are often limits as to how much continuous binary data is permitted. The file is organized into records, where each record contains a portion of the binary data. On MTS, files are divided into records called lines that are limited in length to 32,767 bytes. Nine-track tapes used with MTS also have this limit. The number of records is also included in line 2 of the ERD file header. On systems that allow arbitrarily long binary files, the NRECS value will typically be 1, and the NBYTES parameter will be NCHAN × NSAMP × (BYTES/SAMPLE). When systems have limits restricting the amount of continuous binary data that can be stored, NRECS will be larger than 1 when the total number of bytes in the data part of the file exceeds that limit. For example, a file with 10,000 samples and 20 channels of integer*2 data will require 10,000 × 20 × 2 = 400,000 bytes of storage. Using a maximum record size of 32,760 bytes, 13 records are required. (The record size of 32,760 is the largest size less than the limit of 32,767 that permits an integer number of samples in each record—819 samples in this case.)

When viewed with a text editor, binary data will be unintelligible to most people. It is usually necessary to have a program read the binary data into an array in memory, and then display the data by printing or plotting. On microcomputers, most text editors have trouble

with binary data. Hence it is generally better to use separate files for the header and the data when the data are stored in binary form.

Different computers use different methods to represent numbers in binary form. Different software packages on the same computer may use incompatible representations for storing numbers. All computers can read numbers when they are written out in text form, however. For purposes of transportability, the data portion of an ERD file can also be written in text form. Because there are many ways of writing the text, the header must include the FORMAT statement specifying how the text is written.

The text format is necessary for transporting data in ERD files between different computers. It is also convenient when numbers are typed in manually, or when numbers are to be edited using a text editor. There are penalties for using text representations of numbers, however. First and foremost, the computer must work hard to translate the text numbers into binary form. On MTS, the cost of reading a large file is 10 times greater for numbers in text form rather than binary. On smaller computers, such as the IBM PC or the Apple Macintosh, it will take about 10 times longer to read a file. A second penalty is that text files take up much more space than binary files. To obtain the full precision of floating-point numbers in Fortran, programs require 4 bytes / number in binary, and 13 bytes / number as text. Thus, text files can be three times longer than binary files. On microcomputers with floppy disk drives, disk space can be a serious limitation.

A utility program, SPLIT (see Section 5), can be used to convert ERD files from binary to text and back.

2.3. Nine-Track Tapes

When ERD files are large or numerous, they can be kept on 9-track reel-to-reel tapes. When used at The University of Michigan computer system (MTS), the storage of ERD files on tape is more efficient in terms of space and cost than disk files.

Tape Initialization for Use at MTS

For tapes that will be used on MTS, the following initialization parameters are recommended:

Type:	9-track tapes
Density:	6250 BPI
Labelled:	Yes
Blocking:	Format U (unblocked), Size (max) = 32766

The data format should be binary (integer*2 or real, as appropriate.)

Tape Initialization for Transporting Data Between Computer Installations

When tapes will be sent to, or received from, other institutes, a person from the other installation should be consulted about the initialization settings. The following settings are suggested:

Type:	9-track tapes
Density:	1600 BPI
Labelled:	No
Parity:	Odd
Blocking:	Format FB (fixed block size)
Text:	EBCDIC

The data format should be text, typically using FORMAT = (*n*G13.6) where *n* is a function of the number of channels. The record size must be large enough to hold the longest line from the header, which will be either 80 characters or [8 + 32 × NCHAN], whichever is larger. The block size should be the largest integer multiple of the record size less than 32,767. (Most of the tape length is actually used in the marks separating the blocks. Thus, minimizing the number of blocks allows more files to be stored on the tape.) Usually the data section of the file should use a format so that each record holds 2 or 3 scans (samples of all channels) on each line. (The number of scans / line is specified by parameters in the header.) The SPLIT utility program (see Section 5) can be used to change the format used for text data.

Disk Directories for Tapes

A disk directory is a disk file that contains all of the header information for some (or all) of the ERD files on a tape. From the experience so far, the disk directories could be viewed as essential when dealing with ERD files on tape. There are several reasons that a directory should be created and updated as files are added to a tape.

First, it can be used to update the header information for the tape files. In case some of the labels typed in by a test operator are misspelled, for example, the correct spellings from the directory file would be used in making plots. As another example, the number of samples might not be known when the tape is generated. When the number is discovered, the directory file can be conveniently updated. Often, as the data processing in a project continues, the directories are edited for the purposes of changing units, adding offsets, or standardizing names.

The second reason is that it offers a standard method for automating the processing of selected data on tape. A number of programs exist that use a disk directory to determine which files on a tape are to be processed in a given run. It is very simple to create several directories for the same tape, each containing references to specific subsets of the total number of files on the tape. When the processing program is run, only those files listed in the directory are processed.

A third reason for using a disk-based directory is for improved speed when accessing the tape files. This takes an added importance when processing data interactively, such as when viewing selected data. When the name of a file on tape is specified, the computer normally searches forward for that name to the end of the tape, then rewinds the tape and continues the search from the beginning of the tape to the original position. If the specified file is the next one on tape, this is quite rapid. But if the specified file is the previous one, then the whole tape must be searched. When a disk directory is used, the tape is always

positioned directly to the file of interest. And in the case of the ERD plotter, the common error of naming a file that doesn't exist is immediately detected without searching the tape.

Format of Disk Directories

The format of the disk directory is quite simple, as shown in Table 5. The header from each tape file is echoed exactly, line-by-line. Each header is written in a sector of lines within the disk file. All sectors are allotted a constant number of lines, which should be greater than needed for any of the files. The first line in a sector gives the number of the file on the tape and the name of the files on the tape. The following lines echo the header portion of the tape file. It is this information that can be updated as needed. The organization of the directory file is specified by the first three lines in the files, which give the name of the tape, the number of sectors, and the number of lines in each sector. The first line of the file should begin with the (exact) characters *LONGDIR* so that data-processing programs can recognize the file as having this format.

A utility program called DIR creates a disk directory that references only selected files on the 9-track tape. It is described in Section 5.

Table 5. Summary of Records in a Long Disk Directory.

Sector	*Line (s)*	*Description*
	1	*LONGDIR* followed by tape name. The tape name can be copied from line 2 of a *labelsniff file, and should include the phrase *Vol=.*)
	2	**NS** = no. of sectors in directory (integer)
	3	**NL** = no. of lines in each sector (integer)
1	4	Number of file on tape and name of file on tape. The FORTRAN format is (1X, I4, 1X, A20). (ex: 8 MINN.23.50 ...this is the 8th file on the tape, and it is named MINN.23.50)
	5	*ERDFILEV2.00*
	•	Rest of header for the tape file identified in line 4.
	•	*END*
2	4 + **NL**	Number of another file on tape and its name
	5 + **NL**	*ERDFILEV2.00*
	•	Rest of header for the tape file identified in line 4 + NL.
	•	*END*
NS	4 + (**NS** – 1) × **NL**	Number and name for last tape file
	3 + **NL** × **NS**	Last line of directory file

3. THE ERD PLOTTER

The ERD Plotter is a program on MTS (The University of Michigan computer system) that is used to plot data contained in ERD files. The program is designed to simplify plotting by handling many of the formatting, scaling, labeling, and file-manipulation details that would otherwise be performed manually or with quick-and-dirty programs written for some task at hand. The plotter is written in FORTRAN 77 and can be transported to other computer systems. However, some of the code must be modified to replace primitive graphics routines used on MTS (position the pen, draw a line, etc.) with equivalents for the other computer system. This section contains many details specific to the program as it runs in the MTS environment.

The ERD Plotter, called ERDP, can handle:

- Line plots
- Scatter plots
- Cross plots
- Time histories
- Multiple data sets (on the same axes)
- Log/linear axes
- Grids
- Several axes layouts
- Numerous scaling options, ranging from manual to several fully automatic scaling methods.

Figures 2 through 4 show plots made with ERDP. Figure 2 shows an example of a time history plot. The user selected the file and channel, leaving the scaling and labeling to the plot software. Figure 3 shows summary data in a scatter plot. The ERD file containing this data stored summary measures from different instruments in each "channel," with each "sample" corresponding to a different test site. Figure 4 shows data from several ERD files plotted on log axes. The axis sizes and scale factors were set manually to match other plots, and the labels were selected automatically by the plotter. It combined some of the labels to identify the individual data sets so that they would have unique names. The plotter chose the Test ID label for the entire plot, because the ERD files had the ID in common and nothing else.

The plotter has these additional features:

- "User-friendly" when used interactively
- Convenient access to files on disk and tape, including random-access to MTS tape files
- Filtering of signals (low-pass, high-pass, band-pass)
- Offsetting of signals to separate repeat runs

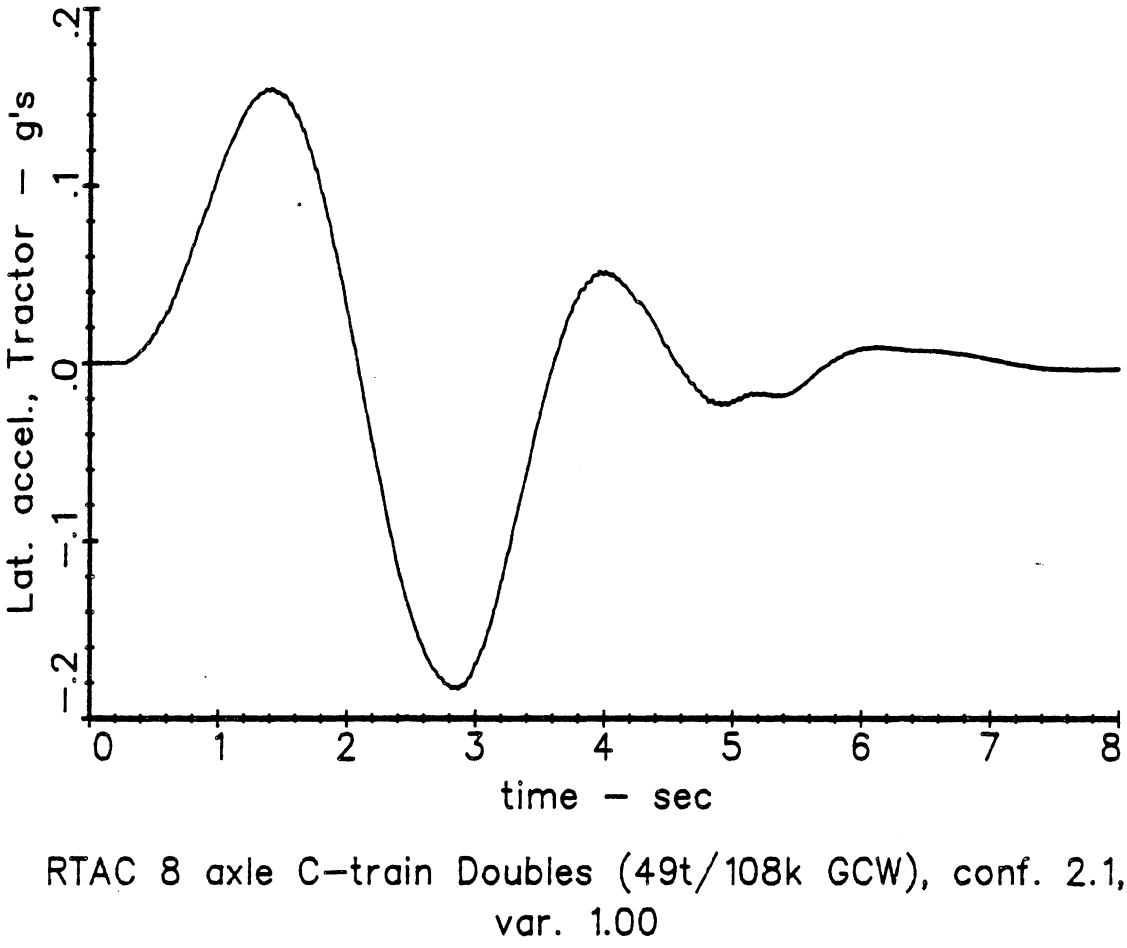


Figure 2. Example time history plot for one variable.

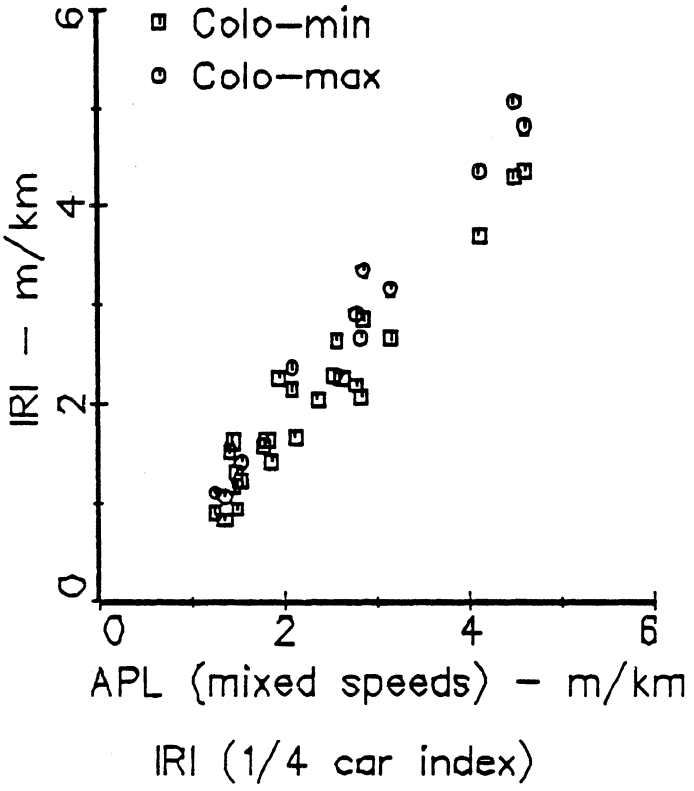


Figure 3. Example scatter plot for two measures.

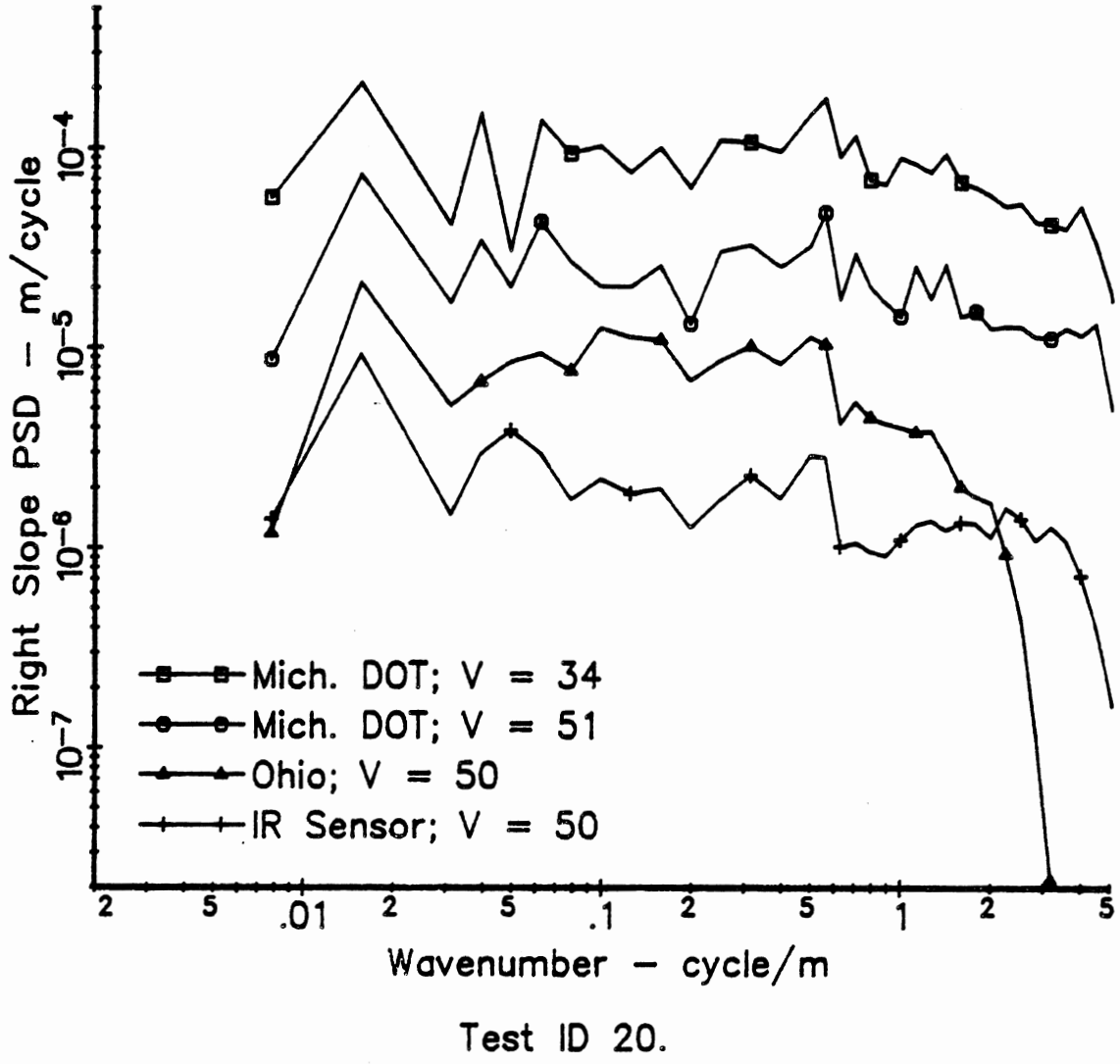


Figure 4. Example log-log plot for measures from several files.

- Automatic labelling of data sets and axes when several data sets are plotted on the same axes
- Automated processing to provide one or more predefined plots for each ERD file

When several data sets are plotted on the same axes, the different data sets can come from the same file or different files, which can be on any mixture of tapes and MTS disk files.

Since the ERD files contain all of the relevant information about the data (names of channels, units, and other categorized information such as test ID number and speed), very little information is needed to look at plots. All that's needed for most applications is the name of the file with the data, and an idea of what should be plotted against what.

The data to be plotted **MUST** be in a file that follows the ERD format.

3.1 Running the Plotter

The plotter and the supporting libraries are contained in the MTS account SVN1. Depending on whether the intent is to look at plots on the screen or to obtain high quality hard copies (CalComp), the procedure used to run the program is modified slightly. The two procedures are illustrated below by example. In these and other examples of MTS dialogs, the statements that you would type at the keyboard to run the program are shown in this type face; `characters and messages from MTS are shown in this type face`, and comments are shown normally.

Procedure to Obtain CalComp Hard Copies Only

(This can be done with any type of terminal—Graphic capabilities are not used.)

```
#create myfile.p

 File "MYFILE.P" has been created.
```

MYFILE.P is the name of the file that will store the plot data until the CalComp has actually made the hard copies. There is nothing special about the name of this file—use anything you want.

```
#sou svn1:.erdp.hc
#$DEBUG svn1:ep.erdp.o+svn1:pl.lib.o+*plotsys
+Ready
```

This source file loads and runs ERDP and the associated libraries under the control of the DEBUG system. (DEBUG allows you to rerun a program while keeping default settings from the previous run. It's also ok to replace the word DEBUG with RUN in the above line if the source file is not used to run the program.)

Running the plotter under DEBUG is more expensive than using the RUN command. If many plots are being made, it is better to use RUN. The prompt character "+" indicates that the DEBUG mode is in effect.

```
+set 9=myfile.p
+Done
```

As plots are generated, they will be stored in the file attached to unit 9.

```
+run
 ERD Plotter, by Mike Sayers, 2-25-86
 How many tapes will you use? {0,1,2, def=0}........
```

Run the ERDP program. See Section 3.2 for the next actions. Each plot is put into the file MYFILE.P as it is generated. Then when all of the plots have been made, the session will conclude with...

```
+User program return.
+Ready
+mts
#r *ccqueue par=myfile.p delivery=tri
```

Leave DEBUG mode, then run the MTS program *ccqueue to generate the hard copies from your file.

```
#Execution begins
   2 plots; plotting requires   105 sec. and    22 in.;      $.18
 Pen was up  38% of the time.
 OK?
 ok
 "STQ4:MYFILE.P     " HAS BEEN PERMITTED "R PKEY=*CCQUEUE".
 PLOT assigned receipt #712165, TRI
  **Leave the PLOT file INTACT through the next PLOT collection time.
#Execution terminated
```

Because *delivery=tri* was specified, the hard copy will be picked up in the regular UMTRI Computer Center run, and should be available at the xerox desk (at UMTRI) in a day or so. The Computer Center also has provisions for mailing Calcomp plots to addresses outside of Ann Arbor.

Dialog to View Plots at a Terminal

(This will require that you use a Tektronix emulator, such as VersaTerm for the Mac.)

```
#%term=versa
```

This tells MTS that you are on a graphics terminal. If you are not using Versaterm, then enter the line: %term=T4014

```
#create myfile.p
 File "MYFILE.P" has been created.
```

This is optional, and is used only if you want CalComp hard copies of some of the plots you will be viewing. MYFILE.P is the name of the file that will store the plot data until the CalComp has actually made the hard copies.

There is nothing special about the name of this file, so use anything you want.

```
#sou svn1:.erdp
#$DEBUG svn1:ep.erdp.o+svn1:pl.lib.o+*ig+*plotsys
+Ready
```

This source file loads and runs ERDP and the associated libraries under the control of the DEBUG system. DEBUG is not needed unless you sometimes make mistakes or want to make plots in one session using different formats. It's also ok to replace the word DEBUG with RUN in the above line and type it directly (instead of using the source file).

DEBUG is more expensive than RUN, so if many plots are being made, it is better to use RUN. The prompt "+" indicates that the DEBUG mode is in effect.

```
+run
 ERD Plotter, by Mike Sayers, 2-25-86
 How many tapes will you use? {0,1,2, def=0} ......
```

Run the ERDP program. See Section 3.2 for the next actions. Each plot will be shown on the screen as it is completed. If you happen to be "stacking" plots, to conserve Calcomp paper in the hard copy, remember that each "stack" is considered by MTS to be a single graphic image. Nothing will appear on the screen until the stack is completed. For example, if the stacking results in five plots being stacked on top of each other, nothing will appear on the screen until the fifth one is completed.

« picture is shown on screen »

```
Blow-up, Redraw, Plot, or Continue?
```

MTS takes control whenever graphics are shown on the screen. Answer *P* (or plot) to add this image to the plot file. If you didn't specify a file (9=...), then MTS will give an error message (no harm done, though) and ask for the name of the file. If this happens, enter *myfile.p* (or whatever name you used). Enter *C* (or continue) to resume with the ERDP program.

3.2 Instructions for Use

After you have gotten to the point of typing RUN, you are using the ERD Plotter (ERDP), together with the MTS DEBUG mode. If you quit the plotter, and are still in the DEBUG mode, and you can restart by typing RUN. Most default values left from the previous run will apply.

When running the plotter, there are usually three levels of control. First, the program is told whether or not tapes are being used. If tapes are being used, then their names are entered, along with the names of the disk directories associated with each tape. This information is required only once in a DEBUG session. At the second level of control, the plotting format is given. All plots made in one run will follow the same rules regarding

scaling, size, and so forth. The various options in the setup basically customize the plotter for the task at hand. After the plot format has been specified, you are in the third level of control, involving the selection of data to be plotted. Further entries are usually limited to the names of the files containing data and the channels of interest in each file.

The second and third levels of control can be automated by the use of plotting templates, so that only the names of the ERD files are needed. Complete automation is allowed if the ERD files are on a tape: all files listed in the directory can be processed with one command.

Conventions for User Input

When running the ERDP program, there is a certain convention for controlling the program.

- All entries are of the question/answer type. ERDP will always ask a question, and provide a default answer that can be accepted by hitting the return key. (One exception: there is no default for the first ERD file name.)
- When file names are requested, ERDP will always check to see if the file exists and repeat the question if the file could not be found.
- Entering *Ctrl-C* for a file name has the same effect as cancelling that option. For example, when ERDP asks for a template file name, entering *Ctrl-C* will indicate that no template file will be used.
- When numbers are requested, spaces are ignored and decimal points are not needed if the number does not have a fractional part.
- When several numbers are to be entered on the same line, then entering only one number and then hitting the return key is the same as entering that number followed by zeros for the following numbers. (If the following numbers are not intended to be zero, then several numbers should be entered, separated by commas.) (Just hitting *Return* has the effect of accepting all of the default numerical values.)
- When numbers are entered, default values of some entries can be accepted by using commas to hold the place. For example, if two numbers are requested, the response *,3* will cause the default value of the first number to be retained, while replacing the second number with the value 3.

Specifying Tapes and Directories

ERDP is presently set up to handle ERD files from up to two tapes, in addition to ERD files stored on disk. It is recommended that a disk directory be generated for each tape, to speed up the time needed to find the file on tape. (The DIR program is described in Section 5.) Also, there is better error checking when a disk directory is used. If tapes are not being used, then this first step is completed by entering *0* when asked for the number of tapes. The following example shows how the program would be used with a single tape mounted.

```
#mount 117-fawn-012 9tp *t* vol=prof9 'prof9'
#117-fawn-012 9tp *t* vol=prof9 'prof9'
#*T* (117-fawn-012): Mounted on T904
```

This is the minimum effort needed to mount a labelled tape. The tape name is assigned by MTS when the tape is submitted, and in this case is *117-fawn-012*. It is to be mounted on a 9-track drive (*9tp*), the pseudo-device name is **T**, the volume name is *prof9*, and the tape ID is also *prof9*. Other pseudo-device names can be used instead of **T**, such as **T1**, **ERD**, or whatever.

```
#sou svn1:.erdp
#$debug svn1:pl.erdp.o+svn1:sc.lib.o+svn1:pl.lib.o+*IG+*PLOTSYS
+Ready
```

Note that this command included **ig**, meaning that all plots will be shown on the screen. Therefore, the terminal must be capable of supporting Tektronix graphics. An output file (9=...) was not specified in this example.

```
+run

ERD Plotter, by Mike Sayers, 2-25-86
How many tapes will you use? {0,1,2 def=0}  1
```

If we had answered *0* or just hit *Return* (for the default), there would be no more questions related to tapes.

```
Pseudo-device name for Drive #1 {def=*T*}    «Return»
```

The tape is in fact named **T**, indicated as the default, so only a *Return* was needed.

```
Disk file with Tape header lines {def=NO DIRECTORY} stq3:pulse2.disk
```

This file had been created earlier. If a disk directory file does not exist for this tape, then just a *Return* would be used. The volume name for the tape is in the disk directory file. If a disk directory is not used, then ERDP will ask for the volume name of the tape.

```
For automated processing, enter name of
template file: {def=*NO TEMPLATE*}  «Return»
```

If this option were to be used, all setup information and channel names would be obtained from the template.

```
Do you want every channel in every file listed in STQ3:PULSE2.DISK
to be plotted automatically? {Y or n, def=n}    «Return»
```

This option is only available when one tape is mounted and a disk directory file is available. In this example, the default (no) is selected. If this option were to be selected, then every file listed in the directory would be

processed. If no template were in effect, then every channel would be plotted from every file.

Specifying the Plotting Format

Once the tape information is provided, ERDP determines how the data in the ERD files will be plotted. There are two ways to provide this information. One is to give the name of an existing setup file. The other is to enter **source** or *Ctrl-C* as the file name, which will cause ERDP to get the information it needs through questions and answers. As features are added to ERDP, the exact sequence of the questions asked in this stage may change. When the setup has been specified, the answers are stored in a file for future use. The default file name is a temporary file, so if the setting will be used again a permanent file name should be entered at that point.

The following is a continuation of the example ERDP session initiated above.

```
Name of setup file (with plotting format info): {def=*SOURCE*}  «Return»
Set size of plot with Window size, Tick interval, or Plot parameters.
Specify Window, Ticks, or Plot {w, t, or p: def=w}  «Return»
```

Accept the default to control the plot scaling by specifying the size of the plotting area. If *T* had been selected, then we would specify scale factors in follow-up questions. If *P* had been selected, then we would enter the nine RANGE array values and the five KEY array values that are used in the PLOT subroutine. This is how scaling options can be specified that are not offered below. (For example, if you want to have auto-scaling on one axis and fixed-scaling on the other, it must be done by specifying the RANGE array elements as described in Section 6.1.)

```
Size of plotting window in inches: {X, Y: def=5.5, 3.5} 5.,3.

Log/Linear Scaling? 0=Lin X, Lin Y, 1=Lin X, Log Y,
                    2=Log X, Lin Y, 3=Log X, Log Y.
Your choice? {0 - 3, def=0}   «Return»

Axis Style?  1=simple axes.              2=axes+plain box
             3=axes+box with tick        4=axes+grid
            -1=axes through origin      -4=axes through origin+grid
             0=no axes
Your choice: {def=1}   «Return»

Scaling options:1 = auto-scaling #1 (best looking)
                2 = auto-scaling #2 (maximum magnification)
                3 = auto-scaling #3 (always include 0 in Y axis)
                4 = manual scaling #1 (specify now for all plots)
                5 = manual scaling #2 (specify later for each plot)
```

```
Your choice: {def=1} «Return»

Do you want to be able to zoom? {Y or n, def=n} «Return»
```

When channel 0 is used for the X axis, ERDP will ask for the range to be plotted if the zoom option was indicated.

```
Scatter plots or line plots? {s or l, def=l}    «Return»

Filter signals? 1 = High-Pass  2 = Lo-Pass  3 = Band-Pass
                0 = none       4 = Decide later
Your choice: {def=0}    «Return»

Plot multiple or single data sets? {m or s, def=s}  m

Space multiple plots vertically by constant offset: {def=0.} «Return»
```

The option to offset the signals is not available when scatter plots are selected or when single data sets are selected. If log scaling had been selected for the Y axis, then a ratio would be requested here instead of an offset.

```
Individual data sets can be identified automatically based on the criteria:
        1 - Rigid body name
        2 - Instrument name
        3 - Speed
        4 - Test ID number
        5 - File title (truncated to 32 characters)
        6 - Channel name (short)
        7 - File name
        8 - Filter setting
        9 - Manual entry (only if all else fails)

        0 - No labelling

You can select any combination, or 0 (No labelling).
The criteria have priority based on the order
that they are entered. Which? {def =  1,  2,  3,  4,  5,  6,  7,  8, 9}
2,3,4,5,9
```

When multiple data sets are plotted on the same axis, ERDP can usually identify the individual data sets and come up with an appropriate plot title. The ERD files can contain a number of labels. ERDP looks for a type of label that is unique for each data set, to identify the individual data sets. To label the entire plot, it looks for a type of label that all of the data sets have in common. Generally, some choices will be preferable to others for reasons outside of the scope of ERDP, so this input allows the user to set priorities and indicate which choices are preferred. ERDP will not even consider using a label unless it is specified here. When several choices are

valid, such as *2,3,4,5,* and *9* in this example, the one listed first (2 in this case) will be used. If the first choice is not valid, the the second choice will be used if it is valid. If none of the choices are valid by themselves, then combinations are tried. If the combinations don't work, and if manual entry (number 9 above) is allowed, then you will be prompted for the needed label.

The labels are all taken from the ERD file. The rigid-body name, the instrument name, the test ID, and the test speed are all optional keywords in the ERD file, while the other names are guaranteed to be available. If any of the files containing data do not use some of these keywords, then the associated labels will not be used by ERDP even if they were selected first in this setup.

In this example, individual data sets will be identified only by instrument name (item 2) if possible. If some of the data sets have the same instrument name, then speed (3) will be tried. Labels related to the rigid-body names, the channel names, and the file names will never be used.

These labelling options are not offered when single data sets are selected.

As ERD files are used in more applications, we can expect that the number of keywords and labelling options will grow. Therefore, the number of options in the current ERDP program may not match the example here. The same logic should apply, however.

```
***DONE!!  Save settings in file: {def=-SETUP} set.demo
```

In this demo, we saved the settings to the file *set.demo*. The next time the plotter is run, we can enter this file name to avoid the preceding questions.

Specifying the Data to Plot

After the tape names (if any) are entered and the plotting format is determined, you make plots by specifying an ERD file and the channel(s) of interest. The input required here depends partly on the files themselves, and partly on the plotting format that is in effect. As a minimum, you must specify the name of the ERD file containing each data set that will be used to create plots. Additional information may also be needed.

ERDP always generates an additional channel (identified as 0) based on the specified *step* parameter from the ERD file. If the file contains time histories, then channel 0 contains the assumed time corresponding to each sample. If the file contains PSDs, then channel 0 contains assumed frequencies. Channel 0 may not always be relevant, but it is always created. If there is only one channel, the only choice for plotting is channel 1 vs. channel 0, and ERDP will go ahead and use those channels without bothering to confirm the obvious. More commonly, the file will contain more than one data channel, and you must indicate which two channels are used for the X and Y axes. The most common choice for the X axis will be channel 0. ERDP asks for the Y channel first, so that the X channel does not have to be reentered every time.

```
Please specify the ERD file containing the data.
Drives: 0 => Disk on STQ4, 1 => Tape PROF9
CTRL-C => no more data, N=next file on tape
File Name for Data Set #1: {def=1:1:I-01-056}  N
```

ERDP assigns a drive number to each tape that was described when the program was started. Number 0 is always the disk drive for the MTS account that you signed on for the current session. This example session was done from the account STQ4, and this is indicated in the above message. Number 1 is the tape that was mounted earlier. If a second tape had been mounted, then the above message would show that 2 => tape2. ERDP always shows the name of the file that will be used as a default if a drive number is not explicitly given for the next file. When entering file names, the drive can always be specified by adding a prefix of two characters to the file name: the drive number and the colon character. In the above dialog, the name of the first file on drive #1 is known from the directory. Files can be specified in many ways, as summarized below:

DISK FILES — consider an MTS disk file called *er.1*. If it is on the current sign-on account (STQ4 in this example), it can be called *0:erd.1* or *STQ4:erd.1*. If the previous file read by ERDP was from disk, then the drive number is not necessary and the name *er.1* can be used. However, if the previous file was on tape, then that tape would be searched for the file *er.1*, probably without success. Files on disk that are under different accounts must always include the account name as a prefix. For example, *ST6U:er.1* is a different file than *0:er.1* in this case. (Note: only files that are properly permitted from other accounts can be accessed.)

TAPE FILES — Only files that are on a tape that was "installed" when ERDP was started can be used. If the default drive is the one containing the tape of interest, then the name of the file can be used by itself. Otherwise, the drive number must be included so that the correct drive will be searched. A file on tape can be identified either by its name (e.g., *1-01-056*) or by a number (e.g., the 11th file on the tape). If the number is known, it can be entered as **n** (e.g., **11**), where the two stars are needed to indicate that we are talking about a file number, and not the name of a file which is a number (e.g., the entry *11* would cause ERDP to search for a file named "11"). One other option exists, which is to get the next file on the tape. This is done by entering *n* or *N* as the name of a tape file. If the entry has more than one character, this will not work. For example, the entry *1:n* would cause ERDP to get the file on drive #1 immediately following the file that was most recently accessed on that tape, while the entry *1:next* would cause ERDP to search the tape on drive 1 for a file named "next."

If there is a disk directory for the tape, the directory is searched for the file name. If it is found in the directory, there may be a pause as the actual tape is rewound or fast-forwarded to the proper position. If the name was not found in the directory, ERDP quickly gives a diagnostic message and asks

again. It is much safer to use a disk directory, because if an incorrect name is entered, the problem is quickly detected and no errors will occur. If an incorrect tape name is entered and there is no disk directory, then the entire tape will be searched (taking several minutes sometimes), and a file will be read anyway. When there is a disk directory, the "n" (next) name will get the next file listed in the directory, which in some cases will not be the next file on the actual tape.

In this example, the next file on tape was indicated.

```
 10/16/85 12:55:12 A-DOLLY PULSE L  90DEG   RUN=  5
This file contains 390 points.
```

If ERDP successfully opens the file, it prints the title and the number of data points so that you have some check that the file you named is in fact the one that you are interested in.

```
There are  14 channels.  Enter "?" to see names, CTRL-C to cancel
Enter Y, X channels {def =  1 - V,  0 - TIME}: ?
```

If the ERD file contains only a few channels, ERDP will show their names. Otherwise, you have to enter a *?* to get a list of the names. Usually, you know the names and won't need this list every time. In this example, a *?* was entered to get the following list.

```
File contains these data channels:
  1  V          {VELOCITY - MPH                          }
  2  STEER      {STEERING WHEEL ANGLE - DEG              }
  3  AYCAB      {CAB LATERAL ACCEL - G'S                 }
  4  YAWCA      {YAW RATE OF CAB - DEG/S                 }
  5  AA1        {ANGLE BETWEEN CAB AND SEMI - DEG        }
  6  AA2        {ANGLE BETWEEN DOLLY + PUP - DEG         }
  7  AA3        {ANGLE BETWEEN SEMI + DOLLY - DEG        }
  8  YAWPU      {PUP TRAILER YAW RATE - DEG/S            }
  9  AYPUP      {PUP TRAILER LATERAL ACCEL - G'S         }
 10  ROLAN      {PUP TRAILER ROLL ANGLE - DEG            }
 11  FXA        {Longitudinal Hitch Force - lbs          }
 12  FYA        {Lateral Hitch Force - lbs               }
 13  FZA        {Vertical Hitch Force - lbs              }
 14  MZA        {Hitch Yaw Moment - in-lbs               }

Enter Y, X channels {def =  1 - V,  0 - TIME}:  aa1
```

Channels can be identified by either name or number. A comma is used to separate the name/number of the channel used for the Y axis from the name/number of the channel used for the X axis. The names used to identify the channels are either the short names taken from the ERD file or else channel numbers. (The short names are the ones shown immediately after the number when ERDP prints a summary.) In this example, it is the

5th channel that is called *AA1*. If only one channel is listed, as in this example, then the X channel is set as 0. We could have also entered *aa1*, *time* or *5*, or *5,0* or *aa1 , 0* or *5, time* for the same effect. ERDP removes any leading spaces, including leading spaces following the comma, and is not sensitive to upper/lower case.

```
Please specify the ERD file containing the data.
Drives: 0 => Disk on STQ4, 1 => Tape PROF9
CTRL-C => no more data, N=next file on tape
File Name for Data Set #2: {def=1:1:I-01-057} aa2
*** That name is not in the disk directory.
File name for data set #2: {def=1:I-01-057} «Return»
```

Oops! Jumped the gun and entered a channel name instead of a file name, resulting in the error message. No problem. The file previously selected is now the default, which was accepted by hitting *Return* on the second try.

```
10/16/85 12:55:12 A-DOLLY PULSE L 90DEG RUN=  5
This file contains 390 points.
There are  14 channels.  Enter "?" to see names, CTRL-C to cancel
Enter Y, X channels {def =  2 - STEER,  0 - TIME}: aa2

Please specify the ERD file containing the data.
Drives: 0 => Disk on STQ4, 1 => Tape PROF9
CTRL-C => no more data, N=next file on tape
File Name for Data Set #3: {def=1:I-01-057}  \
```

We only wanted to plot two channels in this example, so *Ctrl-C* was entered to indicate that ERDP should get on with it. The name *EndofFile* will have the same effect, and can be used when ERDP is run in batch mode.

« picture is shown on screen »

```
Blow-up, Redraw, Plot, or Continue? C

Please specify the ERD file containing the data.
Drives: 0 => Disk on STQ4, 1 => Tape PROF9
CTRL-C => no more data, N=next file on tape
File Name for Data Set #1: {def=1:I-01-057}  \

+User program return.
+Ready
```

Ctrl-C was entered. (It is echoed as the backslash character, \.) Since no files have been opened for this plot, ERDP takes this to mean that we are through, and returns control to MTS in its DEBUG mode. We could start again, possibly to use a different plot format, with the command *run*.

```
+mts
```

#

Instead, we quit the DEBUG system and return to MTS.

3.3 Files Used by the Plotter

ERD Files

The plotter will only read data from ERD files. ERDP will read the data in three forms: binary integer*2, binary real*4 (single-precision floating point), and formatted text. ERDP has some error checking to detect inconsistencies between the data as stored and how it is supposed to be stored. If there are fewer records in the file than indicated in the header, the program will go ahead and proceed normally. But if the number of bytes in a binary record does not match the number that should be there, it will refuse to process the file and will print an error message. If there is not enough memory to read an entire file, ERDP will print a message indicating how much of the file was read.

ERDP makes use of all of the standard header information (gains, offsets, labels) and some of the additional label information. It presently recognizes the additional keywords of GENNAME, XLABEL, XUNITS, SPEEDMPH, RIGIBODY, and TESTID. (See Table 2 in the Subsection 2.1 for the definitions of these keywords.) Other optional lines in the ERD header are ignored.

Setup Files

The plotting format can be stored in a setup file, as described earlier in Subsection 3.2. The easiest way to make a setup file is to run ERDP, entering **source** or *Ctrl-C* for the setup file. When all of the information has been collected, the plotter will ask for a file name. Enter a name for the new file. When the plotter asks the next question, quit the program by using the break key (*Enter* on a Mac). Repeat this process as many times as needed to create the desired setup files.

The first line in the file gives the version of ERDP used to create it, so that the current version of ERDP can recognize old setup files which are no longer valid.

Tape Directory Files

The layout of the directory files is described in Section 2.3. The first eight characters of the first line must either be LONGDIR or SHORTDIR, or else ERDP will not open the file. (The type LONGDIR includes header information for each file, while the type SHORTDIR includes only the file names and the positions of those files on tape.) In addition, the first line must also include the string *Volume=*, which ERDP uses to find the volume name of the tape.

Template Files

Templates can be used to automate plotting. The template is used to bypass the normal control dealing with setup files and channel names. Instead, the template customizes the plotter so that only file names are needed. An entire tape can be processed automatically using a template.

The template file structure is shown in Table 6. The plotter reads the channel names from the template file as if they were typed from the keyboard. Therefore, leading spaces and upper/lower case type are ignored. When only a single channel is listed, the assumed X channel is channel 0.

Template files can be concatenated when running ERDP. This means that when the plotter asks for a file name, you can enter something like

temp.file1+temp.file2+temp.file3

and the plotter will produce plots for all of the templates from the three files. After reading the last line in a template file, ERDP tries to read the next line. If it does not encounter an end-of-file, then it checks to see if the line is the first line of a new template file. If it is, it adds the new plot structures to those that have already been read.

If an ERD file does not contain the channel names listed in a template, that channel pair is skipped and processing continues with other channels in the same plot. Therefore, a single template can be used for many types of ERD files. For example, a template that plots every axle trajectory for a doubles combination would also work without error for a tractor-semitrailer combination. If none of the channels listed for a plot structure are found in the ERD file, then the plot is skipped and processing continues.

There are two types of template files, which differ only in the first line, which will read either TEMPLATE-1 or TEMPLATE-2. A type-1 template is applied to a single file at a time, whereas a type-2 is applied to a list of files. Type-1 is suited for generating standard plots automatically, whereas type-2 is better for comparing data from several files.

Table 6. Layout of a Plot Template File.

Line	*Variable*
1	**VERSION** —*TEMPLATE-1* or *TEMPLATE-2*
2	**NPLOTS** — Number of Plot structures in this file
3	**SETUP.FILE$_1$** — Name of file with setup data for the first plot
4	**ND$_1$** — Number of data sets used in the first plot
5	**CHY$_{1,1}$, CHX$_{1,1}$** — Names of Y, X channels for first data set of first plot
6	**CHY$_{2,2}$, CHX$_{2,1}$** — Names of Y, X channels for second data set of first plot
	•
	•
	•
4 + ND$_1$	**CHY$_{ND_1,1}$, CHX$_{ND_1,1}$** — Names of Y, X channels for last data set of first plot
5 + ND$_1$	**SETUP.FILE$_2$** — Name of file with setup data for the second plot
6 + ND$_1$	**ND$_2$** — Number of data sets used in the second plot
7 + ND$_1$	**CHY$_{1,2}$, CHX$_{1,2}$** — Name of channel for first data set of second plot
	•
	•
	•
	CHY$_{ND_{NPLOTS},NPLOTS}$, CHX$_{ND_{NPLOTS},NPLOTS}$ — Name of Y, X channels for last data set of last plot

4. READING AND WRITING ERD FILES

ERD files are designed to be easily read and written by simple programs written in languages like FORTRAN, BASIC, and PASCAL. In addition, there is a toolbox of subroutines that can be used to conveniently access the information in ERD files from programs written in FORTRAN 77.

4.1 Binary Data

To read and write binary data on the The University of Michigan mainframe computer system (MTS), it is necessary to use the MTS subroutines READ and WRITE. (On other computers, equivalent binary read/write subroutines would be used.) For an example case of 20 channels, 2000 sample/channel, and records sized to contain 100 time steps (scans), the code might be:

```
      REAL XDATA (20, 2000)
      INTEGER*2 NBYTRC
      INTEGER NCHAN /20/, NSCAN /100/, NSAMP /2000/
            •
            •
            •
      DO 10 ISTART = 1, NSAMP, NSCAN
      NBYTRC = 4 * NCHAN * NSCAN
10    CALL WRITE (XDATA (1, ISTART), NBYTRC, 16384, LNUM, 2)
```

When using the READ and WRITE subroutines, the value 16384 sets the appropriate "modifier bits" needed to prevent MTS from deleting blanks from the ends of lines. The toolbox described in Subsection 4.3 contains two subroutines (RDERD and WRTERD) that can also be used to read and write the data portion of the file.

4.2 Description of the ERD File Toolbox

A number of utility subroutines and functions are available that simplify the reading and writing of ERD files. This library is in the MTS file *SVN1:pl.lib.o*. Table 7 lists all of the routines available. (More detailed descriptions are given of the individual subroutines in Subsection 4.3.) All subprograms in the library are written in the FORTRAN 77 language.

Design of the ERD File Toolbox

The subroutines listed in the table are a part of a "toolbox" system, which maintains in memory a copy of the data from the header of the file, and provides an interface to that copy. The interface is accomplished by maintaining pointers into a "heap" of memory containing the data in no particular order. The use of a heap with pointers has the advantage that the toolbox makes efficient use of memory, particularly when several headers are in memory at the same time. Generally, the memory required for several headers is only slightly larger than the memory required for one, because pointers for the

Table 7. Subroutines Dealing with ERD Files.

Data Part of an ERD File

NBIN (NCHAN, NSAMP, NRECS, NBYTES) — compute NRECS and NBYTES.
RDERD (IERD, ARRAY, NDIM, ISTYLE, ICHANS, NCHAN, NSAMP, NRECS, NBYTES, KEYNUM, FRMT, IERR) — read selected data channels.
WRTHED (ARRAY, IFIRST, NCHAN, NSAMP, NRECS, NBYTES, KEYNUM, WRTFMT, IW) — write array into data part of file.

Reading and Writing Headers

POSNTF (ITP, IDSK, NEWFL, INDEX, LSECT) — position tape and disk directory.
RDEND (IR) — read to end of header (Version 2.00 files).
READHD (IHEAD, NCHAN, NSAMP, NRECS, NBYTES, KEYNUM, STEP, KEYOPT, IR) — read a header into memory.
SKIPHD (IR) — skip a header of a file to get to the start of the data.
WRITHD (IHEAD, IW) — write a header from memory to a file.

Getting Data from a Header.

Function GC32 (IHEAD, INDEX, KEYNM) — [char*32] get a char*32 variable.
Function GC8 (IHEAD, INDEX, KEYNM) — [char*8] get a char*8 variable.
Function GC80 (IHEAD, INDEX, KEYNM) — [char*80] get a char*80 variable.
Function IGINT (IHEAD, INDEX, KEYNM) — [integer] get an integer variable.
Function GREAL (IHEAD, INDEX, KEYNM) — [real] get a real variable.

Modifying Data in a Header

DELKEY (IHEAD, KEYNM) — delete data for a keyword from the header.
PC32 (IHEAD, INDEX, KEYNM, STRING) — put a char*32 variable into the header.
PC8 (IHEAD, INDEX, KEYNM, STRING) — put a char*8 variable into the header.
PC80 (IHEAD, INDEX, KEYNM, STRING) — put a char*80 variable into the header.
PINT (IHEAD, INDEX, KEYNM, INUM) — put an integer into the header.
PREAL (IHEAD, INDEX, KEYNM, RNUM) — put a real number into the header.

Duplicating Data from One Header to Another

DUPCH (IHEAD1, IHEAD2, ICH1, ICH2) — duplicate data associated with a channel.
DUPHED (IHEAD1, IHEAD2) — duplicate pointers to data in IHEAD1.
DUPKEY (IHEAD1, IHEAD2, KEYNM) — duplicate pointers to data for one keyword.
DUPOPT (IHEAD1, IHEAD2) — duplicate all optional data in IHEAD1.

Table 7. Subroutines Dealing with ERD Files— continued.

Query Functions.

Function ERRSTR () — [char*60] error message for last toolbox call.
Function IEFERR () — [integer] error code for last toolbox call. (0=cool)
Function ISHORT (IHEAD, STRING) — [integer] channel number based on short name.
Function IWHICH (IHEAD, STRING) — [integer] channel number based on number or short name.

Initializing Headers

CLRHD (IHEAD) — clear all pointers for one header without recovering memory.
CREAHD (IHEAD, NCHAN, NSAMP, NRECS, NBYTES, KEYNUM, STEP, KEYOPT) — create and initialize a header in memory.
INITHD — initialize the toolbox, clear all headers, and restore memory.

Subprograms Used Internally by the Toolbox

CHRINT (BUF, I, IARRAY, N) — read integers from character array.
CHREAL (BUF, R, ARRAY, N) — read real numbers from character array.
Function CMPRST (STRING, KP, N) — [logical] compare string to part of heap.
DEFKEY (KEYNM, KINDAT, N) — install keyword definition.
GETDAT (IHEAD, INDEX, KEYNM, KINDAT, BUF) — get data of type KINDAT.
Function GKEY (I) — [char*8] get keyword associated with internal ID number.
Function ISINHD (IHEAD, KEYNM) — [logical] see if data for a keyword are in header.
Function LOOKUP (KEYNM) — [integer] ID number for KEYNM.
NEWDAT (KEYNM, IHEAD) — set pointers (used by READHD).
Function NITEMS (N, NCHAN, NUMBER) — [integer] items/keyword for type N data.
Function NLEFT () — [integer] number of words left in heap.
Function NWPITM (KINDAT) — [integer] words/item for type KINDAT.
PUTDAT (IHEAD, INDEX, KEYNM, KINDAT, BUF) — put KINDAT data into header.
READH1 (IHEAD, NCHAN, NSAMP, NRECS, NBYTES, KEYNUM, STEP, KEYOPT, IR) — read a version 1.00 header into memory.

different headers point at the same copy of the data. The design also makes it trivial to add new keywords as they are invented.

A disadvantage of the design is that when several headers are in memory at the same time, it is impossible to recover any memory, even when one of the headers is cleared. (This requires "garbage collection," which is not supported within the toolbox.) The only way to recover the memory in the heap is to clear all of the headers at once, using the INITHD subroutine. A method that will lead to trouble when several headers are in use is to clear the old header 1, read a new header 1, then clear the old header 2 to build a new header 2. Instead, clear both headers with INITHD before reading the new header 1.

Reading and Writing ERD File Headers

The subroutine READHD is used to read data from the header of an ERD file into memory. (If the program will be reading several files, it is a good idea to completely clear the heap first using the INITHD subroutine.) After this, all of the information contained in the header can be accessed and modified as needed by using the functions and subroutines listed in the table. The subroutine WRITHD transfers a header from memory into a file. WRITHD writes the entire header at once, so it should be called only after all of the data have been put into the header in memory. Typically, a call to WRITHD is immediately followed by writing the data part of the ERD file.

To create a complete ERD file from scratch, follow these steps:

1. CALL INITHD to completely clear the heap and prevent repeated calls to CREAHD from eventually using up all of the memory.
2. CALL CREAHD to set the basic parameters of the ERD file, such as the number of channels, type of data, and so forth.
3. Put names and other information into the header using the various "Put" subroutines.
4. Open a file for output and attach it to a FORTRAN I/O unit. The subroutine OPNMTS is convenient for this purpose. (OPNMTS is described in Section 7.)
5. CALL WRITHD to write the entire header of the file.
6. Write the data part of the file, using the MTS WRITE subroutine or an equivalent.
7. Close the file.

If the new ERD file is a variation of an existing file, then use READHD instead of CREAHD in step 2.

There are two subroutines that clear data from memory. CLRHD clears all pointers, so that a new header will not contain any data from an earlier header that had the same number. However, if there is another header in memory, it does not release any memory. The INITHD subroutine resets all headers and releases all memory. A program that does not process multiple files will not need to use either of these subroutines. The CREAHD and READHD subroutines will call them automatically if a header has not been cleared or

initialized. However, programs that loop to process an unlimited number of files should use the INITHD subroutine periodically to avoid eventually running out of memory.

Getting Data from a Header

The data in the header are accessed using keywords. Table 4 in Section 2 shows all of the keywords that are presently recognized by the toolbox. The first two lines in the header are the same as shown earlier in Table 1. They are required and do not have explicit keywords; thus, the toolbox assumes the implicit names shown in Table 4. Presently, five kinds of data are put into ERD file headers: integers, floating-point (real) numbers, 8-character names, 32-character names, and 80-character names. The functions IGINT, GREAL, GC8, GC32, and GC80 are used to obtain these five kinds of data.

There are presently four choices for the number of items associated with a given keyword: 1, NCHAN, NLIST, or repeats. Thus all of the GET functions have an argument INDEX, indicating which item on the list is requested. If there is only one data item, INDEX must be set to one. If there are multiple items, the value of INDEX indicates which item is requested. A value of zero is used with IGINT to see how many items are in the header. This is mainly of interest for data in a list of arbitrary length NLIST and for repeated data.

There is one special case that requires a trick. The STEP parameter from line two is a floating point number in the middle of a line of integers. Because the line is defined internally as integer, the GREAL function will return an error if it is given 'LINE 2' as a keyword. This value is normally known from the READHD or CREAHD subroutines, but if there is a reason to get it again, it can be done with the GETDAT subroutine used internally by the other GET functions. When using GETDAT to get STEP, specify the argument KINDAT as 0 (integer) and pass STEP as the argument BUF, where STEP is a real*4 variable.

In some applications, it is convenient to treat the independent variable as an implicit channel 0. The name, units, and offset for the channel may be included, using the optional keywords XLABEL, XUNITS, and XSTART. If INDEX is specified as 0, the GC8 function will return the appropriate short name or units for channel 0. The GC32 function will return the appropriate long name for channel 0, and the GREAL function will return the appropriate offset.

Putting Data into a Header

Data are put into a header in memory with almost the same method used for getting data. Five subroutines are available for putting integer, real, character*8, character*32, and character*80 data into a header based on keyword, index, and the header number. These are PINT, PREAL, PC8, PC32, and PC80.

Be careful never to use these routines to change NCHAN in a header, because the toolbox pointers will no longer be valid.

When putting in data for a keyword that uses an arbitrarily long list, it is necessary to put in the length of the list using PINT with INDEX=0 before putting in any of the data elements. Attempting to put an integer in with INDEX=0 for a keyword that does not have an arbitrarily long list (that is, it has 1, NCHAN, or repeated items) will have no effect other than to set IEFERR to indicate that an error occurred.

In order to change the STEP parameter in line 3, use PINT but pass a floating-point variable or constant for the argument INUM. For example, to set the STEP at 0.25 for header 1, use the statement:

```
CALL PINT (1, 6, 'LINE 2', 0.25)
```

Duplicating Data from One Header to Another

There are several ways to copy data from one header to another. For copying isolated items, a GET routine is used to get the data from a source header and a PUT routine is used to put it into the destination header. If the two headers do not have the same number of channels, this is the only way to move channel names from one to the other, perhaps changing the INDEX along the way. For example, one might get the name of channel 4 from the source header and put it into channel 1 of the destination header.

Four DUP routines are available to duplicate larger amounts of data. DUPCH duplicates all of the data associated with a channel. DUPKEY duplicates all of the data associated with a given keyword. DUPHED will duplicate an entire header. (The previous contents of the destination header are cleared when this routine is invoked.) DUPOPT will duplicate all of the optional data. The DUPKEY and DUPOPT functions will always work for keywords that involve data not tied to channel numbers, such as TESTID, HISTORY, and so forth. If the source and destination headers have the same number of channels, they will also work for keywords such as UNITSNAM that associate a data item with each channel. If the data specified by the keyword in DUPKEY does not exist in the source header, then nothing happens.

The DUP subroutines do not actually copy any data in the heap, and they do not cause any memory to be used. Instead, they set the pointer in the destination header to point at the same data as the pointers for the source header. However, changing the data later with the PUT routines will cause the toolbox to make new copies of the data affected by the PUT routine, so that multiple headers in memory remain independent.

Query and Error Functions

The routines in the toolbox have error checking to guard against simple errors such as a number being out of range, an undefined keyword, or a header not containing the data requested. If an error occurs, a default is returned and a variable within the toolbox is set to a nonzero value. It retains that value until the next toolbox call that has error checking. If there is doubt about the existence of the data, the IEFERR function can be called after calling the GET routines to see if it was successful. If IEFERR is 0, the previous toolbox call was successful. If IEFERR gives a nonzero value, a string containing an associated

message can be obtained by calling the function ERRSTR. The error flag is not reset by calling either of the error display functions.

Two functions are provided that look up the name of a channel. One, ISHORT, returns the channel having the specified short name. The other, IWHICH, is used when several channel names may be in the same string. It accepts either a short name or a number, and allows multiple channels to be separated by commas. Each time it is called, it returns the number for the first channel in the string, and strips that name from the string. Calling it a second time with the same string will get the number of the second channel.

Data Portion of the ERD file

The subroutine RDERD is a one-size-fits-all tool for extracting data channels from an ERD file. It is most useful when only selected channels are to be read, such as when plotting one channel out of 500, or when processing several channels to obtain a summary index. Normally, the function IWHICH or ISHORT would be called first to find the channel numbers of channels whose names are known. Then, RDERD would be called to extract those channels. It will deal with binary integer*2, binary real*4, or text data.

RDERD does not apply offsets and gains to the data. Programs reading data should check for GAINS $\neq 1$ and OFFSETS $\neq 0$.

If the ERD file contains only one channel, or if all of the data will be processed, it will often prove easier to read the data using the MTS READ subroutine, as described in Subsection 4.1, than to set up the ICHANS array to use RDERD.

When writing binary data, it is necessary to set the parameters NRECS and NBYTES that appear in line 2 of the header. The subroutine NBIN will return values that are optimal for use on MTS. The subroutine WRTERD can be used to write the data in an array.

4.3 Subroutines in the ERD File Toolbox

The subroutines and functions listed in Table 7 are described in the rest of this subsection, in alphabetical order. Symbols are used to designate an argument as being:

- $\rightarrow$ input—value is needed by the subroutine and not modified
- $\leftarrow$ output—argument that is assigned a value by the subroutine and which can be undefined going in
- $\leftrightarrow$ both—an argument whose initial value is used, and may be changed by the subroutine.

The compiled subroutines are contained in the MTS file *pl.lib.o* on the SVN1 account; the source code is contained in the file *ef.subs.s* on the same account.

CHREAL (BUF, R, REALS, N)

Convert text data to integers. (Used by NEWDAT subroutine.)

→	BUF	char*32767	character string (array in calling program)
←	R	real	1st number in list
←	REALS	real	1-D array with rest of numbers in list.
→	N	integer	number of numbers in REALS.

CHRINT (BUF, I1, INT4, N)

Convert text data to integers. (Used by NEWDAT subroutine.)

→	BUF	char*32767	character string (array in calling program)
←	I1	integer	1st integer in list
←	INT4	integer	1-D array with rest of integers in list.
→	N	integer	number of integers in INT4.

CLRHD (IHEAD)

Clear pointer and size tables for one of the headers.

→	IHEAD	integer	which header? (1, 2, 3, or 4)

Function CMPRST (STRING, KP, N)

Compare string with capitalized data in heap. (Used by the ISHORT function.)

←	CMPRST	logical	return true if it matches, false if not.
→	STRING	char	string to compare. Must be ALL CAPS.
→	KP	integer	pointer to data in heap. Comparison is from KP+1 to KP+N.
→	N	integer	number of 4-byte words to compare.

CREAHD (IHEAD, NCHAN, NSAMP, NRECS, NBYTES, KEYNUM, STEP, KEYOPT)

Create new header with defaults for the required data.

→	IHEAD	integer	which header? (1, 2, 3, or 4)
→	NCHAN	integer	no. of channels in ERD file.
→	NSAMP	integer	no. of samples in data part of file.
→	NRECS	integer	no. of records in data part of file.
→	NBYTES	integer	no. of bytes/record in data part of file.
→	KEYNUM	integer	key indicating format of data part of file.
→	STEP	real	sample interval.
→	KEYOPT	integer	key used in some data processing programs.

DEFKEY (KEYNM, KINDAT, N)

Install a new keyword into the toolbox. (Used by INITHD.)

→	KEYNM	character*8	name of keyword.

→	KINDAT	integer	kind of data. 0 = integer data, 1= real data, 2= char*8 data, 3 = char*32 data, 4 = char*80 data.
→	N	integer	how many items/keyword? 1=1/file, 2=1/channel, 3=NLIST, where NLIST is the first item on the list and NLIST items follow, 4=Repeated.

DUPKEY (IHEAD1, IHEAD2, KEYNM)

Duplicate data for a keyword from a source header to a destination header. If the data are of type-2 (1/channel) both headers must have the same number of channels.

→	IHEAD1	integer	source header.
→	IHEAD2	integer	destination header.
→	KEYNM	character*8	keyword.

DUPHED (IHEAD1, IHEAD2)

Duplicate header by copying pointers.

→	IHEAD1	integer	source header.
→	IHEAD2	integer	destination header.

DUPOPT (IHEAD1, IHEAD2)

Duplicate all optional data between headers by copying pointers.

→	IHEAD1	integer	source header.
→	IHEAD2	integer	destination header.

Function ERRSTR ()

Get string with message explaining errors involving keywords, the heap, etc.

←	ERRSTR	character	error message for last toolbox subroutine.

GETDAT (IHEAD, INDEX, KEYNM, KD, BUF)

Get item of data. (Used by functions GC8, GC32, GC80, IGINT, GREAL.)

→	IHEAD	integer	which header? (1, 2, 3, 4)
→	INDEX	integer	index of data item for that keyword.
→	KEYNM	char	keyword
→	KD	integer	kind of data. (Used for error checking.)
←	BUF	?	returned data.

Function GC8 (IHEAD, INDEX, KEYNM)

Get 8-character name if it is in the heap.

← GC8	character*8	character string.
→ IHEAD	integer	which header? (1, 2, 3, 4)
→ INDEX	integer	index of data item for that keyword.
→ KEYNM	character	keyword

Returns XUNITS or XLABEL names if channel 0 is requested for UNITSNAM or SHORTNAM.

Function GC32 (IHEAD, INDEX, KEYNM)

Get 32-character name if it is in the heap.

← GC32	character	character string.
→ IHEAD	integer	which header? (1, 2, 3, 4)
→ INDEX	integer	index of data item for that keyword.
→ KEYNM	character	keyword

Returns XLABEL name if channel 0 is requested for LONGNAME.

Function GC80 (IHEAD, INDEX, KEYNM)

Get 80-character name if it is in the heap.

← GC80	character	character string.
→ IHEAD	integer	which header? (1, 2, 3, 4)
→ INDEX	integer	index of data item for that keyword.
→ KEYNM	character	keyword

Function GREAL (IHEAD, INDEX, KEYNM)

Get real*4 number if it is in the heap.

← GREAL	real	value of real data.
→ IHEAD	integer	which header? (1, 2, 3, 4)
→ INDEX	integer	index of data item for that keyword.
→ KEYNM	character	keyword

Returns STEP or XSTART parameters if channel 0 is requested for gain or offset.

Function IEFERR ()

Get integer error flag. The variable IERROR is in the common block /EFTERR/. This function provides that value to subroutines that do not use that common block.

← IEFERR	integer	error code for last toolbox subroutine.

Function IGINT (IHEAD, INDEX, KEYNM)

Get integer if it is in the heap. Special case for INDEX = 0 returns number of items for that key.

← IGINT	integer	value of integer data.
→ IHEAD	integer	which header? (1, 2, 3, 4)
→ INDEX	integer	index of data item for that keyword.
→ KEYNM	char	keyword

INITHD

Define the tables used by the toolbox. (This subroutine contains the only code that needs to be changed if keywords are added to the toolbox.)

Function ISHORT (IHEAD, SHORTN)

Look up short name and return channel number. XLABEL is treated as channel 0.

← ISHORT	integer	channel number, if > 0. -2 → channel name not recognized
→ IHEAD	integer	which header? (1, 2, 3, 4)
↔ KEYBUF	char	character buffer. This function strips the beginning, up to the first comma or the 20-th position.

Function IWHICH (IHEAD, KEYBUF)

Find out which channel was selected and is in the string KBUF. Strip that part out of KEYBUF so that the function can be called for the remainder.

← IWHICH	integer	channel number, if > 0. -1 → channel number out of range -2 → channel name not recognized
→ IHEAD	integer	which header? (1, 2, 3, 4)
↔ KEYBUF	char	character buffer. This function strips the beginning, up to the first comma or the 20-th position.

When IWHICH returns, STRING will be shifted to the first comma or by 20 characters, whichever is smaller. (E.g., Input = ' 3, 4', output = ' 4'.) Thus, the subroutine can be called several times in the event that several channel names are contained in the same string and separated by commas. The subroutine will work either with numbers or names to specify a channel.

Function LOOKUP (KEYNM)

Look up name of key in list.

← LOOKUP	integer	index into keyword table used by the toolbox.
→ KEYNM	character	keyword

NBIN (NCHAN, NSAMP, NRECS, NBYTES)

Compute semi-optimal numbers needed for (real) binary data for use on MTS.

→ NCHAN	integer	number of channels.
→ NSAMP	integer	number of samples.

← NRECS	integer	number of records with binary data.
← NBYTES	integer	number of bytes / record.

NEWDAT (KEYNM, IHEAD)

Add data to the heap used by the toolbox. (Used by the READHD subroutine.)

→ KEYNM	character	keyword
→ IHEAD	integer	which header? (1, 2, 3, 4)

Function NITEMS (N, NCHAN, NUMBER)

How many items for this keyword?

← NITEMS	integer	number of items.
→ N	integer	element from NXPECT table.
→ NCHAN	integer	number of channels, used only if it's type 2.
→ NUMBER	integer	first word of record, used only if it's type 3.

Function NWPITM (KD)

What kind of data? (How many words/item?)

← NWPITM	integer	number of 4-byte words / item.
→ KD	integer	value from the KINDAT table.

PC8 (IHEAD, INDEX, KEYNM, STRING)

Put character*8 string into a header.

→ IHEAD	integer	which header? (1, 2, 3, 4)
→ INDEX	integer	index into list of data for this keyword
→ KEYNM	character	keyword
→ STRING	character*8	string of 8 characters to put into the header.

PC32 (IHEAD, INDEX, KEYNM, STRING)

Put character*32 string into a header.

→ IHEAD	integer	which header? (1, 2, 3, 4)
→ INDEX	integer	index into list of data for this keyword
→ KEYNM	character	keyword
→ STRING	character*32	string of 32 characters to put into the header.

PC80 (IHEAD, INDEX, KEYNM, STRING)

Put character*80 string into a header.

→ IHEAD	integer	which header? (1, 2, 3, 4)

→	INDEX	integer	index into list of data for this keyword
→	KEYNM	character	keyword
→	STRING	character*80	string of 80 characters to put into the header.

PINT (IHEAD, INDEX, KEYNM, INTNUM)

Put an integer value into a header.

→	IHEAD	integer	which header? (1, 2, 3, 4)
→	INDEX	integer	index into list of data for this keyword
→	KEYNM	character	keyword
→	INTNUM	integer	number to put into the header.

POSNTF (ITP, IDSK, NEWFL, INDEX, LSECT)

Position tape and read to proper line in disk directory file.

→	ITP	integer*4	FORTRAN device number of tape. If ITP < 0, then tape is ignored.
→	IDSK	integer*4	FORTRAN device number of disk directory. If IDSK < 0, then disk directory is ignored.
→	NEWFL	integer*4	file number for tape (e.g., 9^{th} tape file).
→	INDEX	integer*4	sector number in directory. (E.g., 4^{th} sector in disk directory.)
→	LSECT	integer*4	number of lines/sector in disk directory.

This subroutine might be used after an ERD file on tape has been selected, before reading any data. Note that it can used to position the tape alone, the disk directory alone, or both. Warning: this subroutine needs to be modified some day. It is inefficient and can be expensive when dealing with large directories.

PREAL (IHEAD, INDEX, KEYNM, RNUM)

Put a floating point numerical value into a header.

→	IHEAD	integer	which header? (1, 2, 3, 4)
→	INDEX	integer	index into list of data for this keyword
→	KEYNM	character	keyword
→	RNUM	real	number to put into the header.

PUTDAT (IHEAD, INDEX, KEYNM, KD, BUF)

Put data of a given type into the heap. Insert if possible, otherwise add to end of heap.

→	IHEAD	integer	which header? (1, 2, 3, 4)
→	INDEX	integer	index into list of data for this keyword
→	KEYNM	char	keyword
→	KD	integer	kind of data. (Used for error checking.)
→	BUF	?	provided data.

RDEND (IERD)

Continue reading from header to END, without interpreting the information in it.

→	IEND	integer	FORTRAN device number of ERD file.

RDERD (IERD, ARRAY, NDIM, ISTYLE, ICHANS, NCHAN, NSAMP, NRECS, NBYTES, KEYNUM, FRMT, IERR)

Read the numerical data portion from an ERD file.

→	IERD	integer	FORTRAN device number of ERD file.
←	ARRAY	real*4	the array that is filled with the data.
→	NDIM	integer*4	one of the two dimensions of ARRAY.
→	ISTYLE	integer*4	This has a double meaning that tells the subroutine the way that ARRAY is dimensioned in the calling program. If ISTYLE ≤ 0, use ARRAY (NDIM, NSAMP) If ISTYLE > 0, use ARRAY (ISTYLE, NDIM)
→	ICHANS	integer*4	array containing channels that will be returned. ICHANS(i) is the channel number that will be put into the the i^{th} channel in ARRAY. There must be (at least) NDIM elements in the ICHANS array. If any elements of ICHANS are < 1 or > NCHAN, then the corresponding channels in ARRAY are left alone.
→	NCHAN	integer*4	number of channels in ERD file.
↔	NSAMP	integer*4	number of samples in ERD file. (Note that this variable may be changed by RDERD if the file ends prematurely.)
↔	NRECS	integer*4	number of records in ERD file. (Note that this variable may be changed by RDERD if the file ends prematurely.)
→	NBYTES	integer*4	number of bytes per record in the ERD file.
→	KEYNUM		integer*4 key indicating the format of the numerical data in the ERD file: 0 → integer*2, binary 1 → real*4, binary 5 → real*4, text
→	FRMT	character*32	format used to read data from ERD file if KEYNUM = 5. (Otherwise, FRMT is not used).
←	IERR	integer*4	Error code, sum of following: 0 = No problems 1 = Number of samples returned was less than NSAMP 2 = Wrong number of bytes read in a record 4 = Wrong number of records was read (too few) 8 = Bad KEYNUM value (only 0,1,or 5) - no data read 16 = ERD parameters don't check, but data were read 32 = ERD parameters don't check - no data read

READHD (IHEAD, NCHAN, NSAMP, NRECS, NBYTES, KEYNUM, STEP, KEYOPT, IR)

Read header from file and put into memory for access with the toolbox.

→	IHEAD	integer	which header? (1, 2, 3, 4)
←	NCHAN	integer	no. of channels in ERD file.
←	NSAMP	integer	no. of samples in data part of file.
←	NRECS	integer	no. of records in data part of file.
←	NBYTES	integer	no. of bytes/record in data part of file.
←	KEYNUM	integer	key indicating format of data part of file.
←	STEP	real*4	sample interval.
←	KEYOPT	integer	key used in some data processing programs.
→	IR	integer	FORTRAN unit for input file.

READH1 (IHEAD, NCHAN, NSAMP, NRECS, NBYTES, KEYNUM, STEP, KEYOPT, IR)

Read header from version 1.00 file and put into memory for access with the toolbox. (Used by READHD when version 1.00 file is encountered.)

SKIPHD (IR)

Skip header from file, so that next read will be the beginning of the data.

→	IR	integer	FORTRAN unit for input file.

WRITHD (IHEAD, IW)

Write header from memory into a file.

→	IHEAD	integer	which header? (1, 2, 3, 4)
→	IW	integer	FORTRAN unit for output file.

WRTERD (ARRAY, IFIRST, NCHAN, NSAMP, NRECS, NBYTES, KEYNUM, WRTFMT, IW)

Write data part of ERD file, using WRITE subroutine on MTS for binary write.

→	ARRAY	real/int	array with numerical data to be stored in file.
→	IFIRST	integer	first sample to store. Ex: IFIRST = 3 → first 2 samples in each channel are not written.

(The next 5 parameters are from line 2 in the header of an ERD file)

→	NCHAN	integer	number of channels *and* 1st dimension of ARRAY.
→	NSAMP	integer	number of samples/channel to be written.
→	NRECS	integer	number of records to be written.
→	NB	integer	number of bytes/record for binary data.
→	KEYNUM	integer	code for data storage type. 0=int, 1=real, 5=text
→	WRTFMT	char	FORTRAN FORMAT, used only when KEYNUM=5

→ IW integer FORTRAN i/o unit number for writing.

4.4 Programming Notes

The common block called /EFTTAB/ is used to store headers in memory. Table 8 defines the arrays, parameters, and variables stored in the /EFTTAB/ common block. The array KEYWRD stores the names of the keywords recognized by the toolbox; the arrays KINDAT and NXPECT define the type of data associated with each keyword; the array KPOINT contains pointers into a heap; and NWORDS contains the number of 4-byte words stored in the heap. The heap itself is contained in three arrays, which are declared as having types: integer, real, and character*4. The three arrays, called INT4, REAL4, and CHAR4, respectively, refer to the same memory location due to the use of an EQUIVALENCE statement. That is, INT4(10) refers to the same 4-byte section of computer memory as do REAL4(10) and CHAR4(10). The number of keywords recognized by the toolbox is called NKEYS, and the pointer to the last element in the heap that has data is called LAST.

The INITHD subroutine sets up the KEYWRD, KINDAT, and NXPECT arrays which define all of the data recognized by the toolbox. As new keywords are invented, this routine can be modified to add new definitions using the DEFKEY subroutine. INITHD also clears all pointers, by setting every element in the KPOINT and NWORDS arrays to zero, and by setting LAST to zero.

Data in memory are referenced by a keyword, a header number, and an index. The keyword and header number correspond to the two dimensions of KPOINT and NWORDS. Consider keyword number I and header number J . The data are stored in the heap beginning at position KPOINT (I, J) + 1 and continuing to position KPOINT (I, J) + NWORDS (I, J). To get the data associated with an index, the NWPITM function is used to determine how many words are used per item. When index=1, the data begin at the first position. When index=2, they start NWPITM words further into the heap.

Repeatable data (e.g., HISTORY) are located by a pair of pointers. The first pointer points to a second pointer, which in turn points to the data. The first pointer is in the KPOINT array. It points to a location in the heap, that contains MAXHIS consecutive pointers. The first of these points to the data for the first occurrence, the second points to the data for the second occurrence, and so forth. The number of items is determined by dividing the number of words in NWORDS (I, J) by the number of words per item.

Size limits of the toolbox are defined by the parameters MAXHEP, MAXKEY, MAXHED, and MAXHIS. MAXHEP is the size of the heap. On MTS, this is set to 50,000 4-byte words (200,000 bytes) to accommodate ERD files that can contain up to 500 channels. When fewer channels are used, a smaller size is appropriate. MAXKEY is used to dimension the arrays, and must be greater than the maximum number of keywords recognized by the toolbox. MAXHED is the maximum number of headers that can be in memory at the same time, and is presently set to four. MAXHIS is the maximum number of repeats allowed for repeatable keywords, and is presently set to ten.

Table 8. Definitions for the Data Structure Used in the ERD File Toolbox.

FORTRAN Name	*Description*
NKEYS	Number of keywords installed in toolbox.
KEYWRD (I)	Name of I-th keyword installed in toolbox.
KINDAT (I)	Kind of data associated with I-th keyword. 0 → integer. 1 → floating-point. 2 → character*8. 3 → character*32. 4 → character*80.
NXPECT (I)	Number of data elements to expect for I-th keyword. 1 → 1 element for an entire file. 2 → 1 element for each channel. 3 → N elements, where N is the first number in a list. 4 → repeats, 1 for each time the keyword appears in the file.
KPOINT (I, J)	Pointer to beginning of data associated with keyword I and header J. The data begin at position KPOINT (I, J) + 1 in the heap.
NWORDS (I, J)	Number of 4-byte words reserved in heap for data associated with keyword I and header J. For numbers in a list or repeating data, N = NWORDS (I, J) / NWPITM (KINDAT (I)) where NWPITM is a function in the toolbox.
LAST	Pointer to last 4-byte word used in heap.
INT4 (K)	Integer interpretation of K-th element of heap.
REAL4 (K)	Floating-point interpretation of K-th element of heap.
CHAR4 (K)	Character*4 interpretation of K-th element of heap.
MAXHEP	Maximum size of heap (4-byte words).
MAXKEY	Maximum number of keywords that can be installed.
MAXHED	Maximum number of headers that can be in memory at the same time.
MAXHIS	Maximum number of times a keyword can be repeated in a file.

5. UTILITY PROGRAMS

There are several stand-alone programs on the *svn1* account on MTS that are useful for manipulating ERD files. All of the programs make use of subroutines that are in libraries also stored on the *svn1* account. To run any one of these programs, a run command must be used that includes the names of all files containing subroutines that are used by the program. To make this a little easier, each utility program can be invoked using a source file with the command:

$sou filename

If any additional information is needed (e.g., a file name) the programs will ask for it.

All of the source files begin with a period, so that they can be listed easily using the MTS $filestatus command:

$f svn1:.?

```
.COPYERD  .DIR      .ERDP     .ERDSUM   .F2E        .FP
.FUNCP    .SPLIT
```

.COPYERD — Copy selected ERD files from one tape to another and update headers.

As input, this program requires a tape and a disk directory for that tape with copies of the headers of the ERD files. As output, it uses a second tape. The output files have the data from the source tape and the header information from the disk directory. When the information in the header of an ERD file needs to be edited, the changes can be made in the copy on disk, and then this program can be run to create an updated copy on tape.

.DIR — Create a disk directory for a tape with ERD files.

As input, this program requires a tape containing ERD files and also a disk file with a list of the files to put into the directory. (The input disk file is intended to be the output of the MTS program **Labelsniff*.) The output is a directory, as defined in Table 5 in Section 2.

To create the directory file,

1. Run **Labelsniff* for the tape with the ERD files, routing the output into a file. Unit 0=tape, sprint=output file.
2. Edit the **Labelsniff* file, deleting the lines corresponding to any files that should not be included in the directory. Do not delete the first 5 lines of the file, and do not change any of the lines in the first 6 column positions.
3. Run the utility program by typing: *sou svn1:.dir*

.ERDSUM — Print a summary of the ERD files referenced in a disk directory.

.F2E — Convert two files from an IBM PC into an ERD file on MTS.

Data measured with an UMTRI digital data-acquisition system can be routinely sent to MTS and put into the ERD file format. The data are generally stored on a 3M tape cassette, for a tape recorder attached to an IBM PC. First, a program on the IBM PC is used to create two files that can be transmitted to MTS using the Kermit file transfer protocol. After the files are transmitted to MTS, the F2E program is used to merge the two files into a single ERD file.

3M is a utility program for viewing data on the IBM PC acquired using an UMTRI system. Among its capabilities, it prepares the disk files that are later sent over to MTS. To create the files on the IBM PC, run the 3M program. (To avoid conflicting with various system configurations, the IBM should be booted with a standard MS DOS 2.0 disk. Next, put in the program disk and type *3M*.) To generate the files, follow these steps:

1. The program begins with a menu of options. Select TRANSFER FILES TO FLOPPY.
2. The program asks for a tape ID. This is a single alphabetic character which will begin all of the file names. For example, if the ID is *C* the files will have names such as *C-10-041*.
3. The program asks for a segment number. This will also be used in the names of the files generated. In the above example, the segment is *10*.
4. The program will prompt for a range of files. They are referenced by the numbers on the tapes. For example, an entry of *34 ... 50* would cause the program to process files 34, 35, ... 50 on the tape.
5. Put in formatted floppy disks when requested by the program. When preparing many files, it is a good idea to prepare several blank formatted floppy disks before running 3M.

The 3M program creates two files for each original tape file, with the same name but different MS DOS extensions: *.TXT* and *.DTA*. The file with the *.DTA* extension contains integer*2 binary data, and the file with the *.TXT* extension contains an abbreviated version of the header of an ERD file.

The files are next transmitted to MTS. To do this, connect an IBM PC to MTS using a modem or the SCP connection at UMTRI. Run the KERMIT communications program on the PC, using version 2.29 or greater. When transferring files with KERMIT, different protocols are required for binary and text files. Therefore, it is usually best to transfer all of the text files (the ones with the *.TXT* extension) together, and then send all of the binary files (the ones with the *.DTA* extension). To do this, follow these steps:

1. Run the program KERMIT on the IBM PC. With the program running, the prompt should appear: MS-KERMIT.

2. Type *SET PARITY EVEN*
3. Type *SET BAUD 9600*
4. Type *C*. This should establish a connection to MTS. Sign on to the MTS account where the files are to be transferred.
5. Run the MTS program **Kermit*. The prompt should appear UM-KERMIT.
6. Type *SERVER* to put MTS into the server (automatic) mode, and escape to the local (MS DOS) Kermit by typing *Ctrl-] C*.
7. Send all of the text files to MTS using the *SEND* command. For example, the command *SEND B:*.TXT* would send all of the *.TXT* files on the floppy disk in drive B to MTS. When all of the files have been successfully received by MTS, put in the next floppy and repeat.
8. After all of the text files have been sent, enter the command *FINISH*. This brings MTS out of the server mode. Then type *C* to reconnect with MTS.
9. *Kermit* (on the MTS end) must be configured to transfer binary files. Type *SET FILETYPE BINARY*
10. Type *SET BINARY-BLOCKSIZE 32000*
11. Type *SERVER*
12. Send all of the binary files, using the *SEND* command as was done in step 7 for the text files. The binary files all use the extension *.DTA*.
13. Type *FINISH* and resume the connection to MTS with the command *C*. The files are all on MTS now, where they can be merged using the F2E program.

The text files from the IBM PC contain the number of channels, the number of samples, the gains and offsets for each channel, a 5-character name for each channel, and a 5-character name for the units used for each channel. The F2E program looks up the 5-character names in a dictionary file to obtain 8-character short names, 32-character long names, rigid-body names, and generic names for use in building the ERD file. The file *SVN1:.ut.f2e.tab* contains the dictionary for some channel names that have been used before. The file with the table of names should be edited to include all of the names that might be encountered. If the files being transformed include names that are not in the table, they will be given generic names like "Channel 11."

The F2E program is fairly simple to run. When it asks for the name of a pair of files to transform, the name should not include the IBM extension. For example, suppose the files transferred from the IBM PC are *C-10-043.TXT* and *C-10-043.DTA*. The name that should be entered is *C-10-043*. The F2E program will then create an ERD file called *ER.C-10-043*.

The program can put the ERD files on a tape, and it can process a list of file names from a text file. If the files are put onto tape, it automatically goes to the end of the tape so as not to overwrite any existing data.

.SPLIT — Splits the ERD file into smaller files; converts data format from binary to text.

As input, this program has one ERD file. As output, it has one or more ERD files. SPLIT serves several functions:

- It can divide an ERD file into smaller files containing contiguous portions of the numercial data. For example, a file with test data sampled over 10 seconds can be split into three files covering 0-2, 2-7, and 7-10 seconds.
- It can extract channels from a file. The list of channels to extract can be stored in a text file to partially automate the process.
- It can convert between binary and text data. When the output file is specified as having text data, various formatting options are available.

If the output files are put onto tape, SPLIT automatically goes to the end of the tape so as not to overwrite any existing data.

6. PLOTTING SUBROUTINES

The plotting package is designed to generate engineering plots in "final form" for use in reports and papers, as painlessly as possible. The package consists of specialized subroutines, which are in turn called by a single subroutine named PLOT. The PLOT subroutine is called from a FORTRAN program, performing all of the plotting-related computations for data provided to it. It uses some of the subroutines in the **Plotsys* library on the mainframe computer system at The University of Michigan, and in addition, a number of additional library subroutines to add capabilities beyond those available in **Plotsys*.

There are also three stand-alone programs that act as front-ends for PLOT, to allow interactive use of the subroutine:

- *ERDP* plots data from ERD files and is described in Section 3.
- *FP* plots data contained in text files having arbitrary layout and is described below in Subsection 6.2. It is the predecessor of the ERDP program. FP requires entering scaling options, labels for the axes, and the format used in the text file.
- *FUNCP* is similar to FP except that it plots data that is generated using a mathematical function. It is described in Subsection 6.3.

The PLOT package produces line plots, scatter plots, and line plots with symbols to identify the lines. The axes are rectangular and can be log or linear. Up to 30 data sets can be plotted together, and eleven of those can be individually identified. Although full control of the plotting parameters is allowed, the package has thorough auto-scaling features that will usually produce the best looking plot possible. (The main reason for using the manual scaling is to produce plots that match the scaling of other specific plots.) Several priorities are available for the auto-scaling logic. Usually, it tries to show the data with the greatest detail possible while making nice looking axes. Two other options are to always include zero as a reference, or to show absolutely the maximum detail possible.

A plot can be considered to contain a few basic elements:

- X axis, including tick marks and label
- Y axis, including tick marks and label
- Axis mode (several choices)
- Title line(s)
- Data
- Symbol key

At the very minimum, it is necessary to provide data, a title, the axis mode, and if axes are shown, labels for each axis. With the additional specification of size and labels for multiple data sets, everything needed for most applications is taken care of.

At the maximum, the scaling of the axes can be specified completely to obtain ugly plots. (For even greater aesthetic atrocities, the axis/grid/printing subroutines in the packages can be used directly.)

6.1 The Plot Subroutine

Data Structure and Specification

The data to be plotted are described by these parameters.

NDSETS Number of data sets to be plotted. (1 - 30)

NPTS(i) Number of X-Y data pairs in i-th data set. (i=1...NDSETS) The total number of points is thus:

$$N_{total} = \sum_{i=1}^{NDSETS} NPTS(i)$$

XYDATA(i,j) XYDATA is a two-dimensional array that contains all of the X and Y values that get plotted. XYDATA(1,j) = X value for j^{th} point, XYDATA(2,j) = Y value for j^{th} point. In sequence, it contains all of the X-Y pairs for the first data set, then all the pairs for the second data set, etc.

Summary of Plotting Options

The options are specified by parameters contained in three arrays: the RANGE array which contains 9 scaling-related parameters, the KEY array, which contains 5 mode-related parameters, and the NSYMB array, which tells how points from each data set will be represented. The KEY and NSYMB parameters are fairly simple, and are described below. The RANGE parameters interact, and are described in the next section, according to the various modes available using the PLOT package.

KEY array

(1) LOG-LIN key. (0=lin-lin, 1=log Y, 2=log X, 3=log-log)

(2) Axis Mode Key. (0=no axes; +1=axes intersecting at lower-left corner of plotting area; -1,-2,-3=axes going through origin; +2=axes with ticks at left and bottom, plain lines at top and right; +3=plot area surrounded by box with tick marks, +4=axes at left and bottom, full grid; -4=axes through origin, full grid)

(3) Location of Symbol Key. (0=no key is shown; 1=lower-left corner of area; 2=upper-left, 3=upper-right, 4=lower right, 5=above plot area; 6=right side of plot area; other value (< 0, > 6)=where it will fit best.

(4) Stacking option on Calcomp paper. (0=don't stack, anything else=stack plots)

(5) Key used for semiautomatic scaling. Only applies when XMAX=XMIN or YMAX=YMIN. 0=no hassle. +=User is told max, min values and must enter replacements. -=User is told max, min values for information only.)

NSYMB array

There is one element for each data set used. NSYMB(i) refers to the i^{th} data set. The data points can be shown in four different ways:text14

NSYMB(i) = 0	*Line Plot*: X-Y points in i-th data set are connected with straight lines. No symbols are used to identify individual points.
NSYMB(i) < –10000	*Thick Line Plot:* X-Y points in i-th data set are connected with thick straight lines, with no symbols shown for individual points. This is done by making four repeated plots, spaced slightly so that the lines will slightly overlap using the normal pen size.
NSYMB(i) > 0	*Line Plot with Identifier:* X-Y points in i-th data set are connected with straight lines. An identifying symbol (square, triangle, etc.) is used to identify every NSYMB(i)-th point.
NSYMB(i) < 0	*Scatter plot:* every NSYMB(i)-th point in the i-th data set is shown by a symbol. The symbols are not connected by lines.

RANGE array (size and scaling parameters)

(1)	XMAX	Max value covered by X axis. (Engineering units)
(2)	XMIN	Min value covered by X axis. (Engineering units)
(3)	YMAX	Max value covered by Y axis. (Engineering units)
(4)	YMIN	Min value covered by Y axis. (Engineering units)
(5)	XLEN	Length used to size X axis. (inches)
(6)	YLEN	Length used to size Y axis. (inches)
(7)	XTICK	Interval between major tick marks. (Engineering units)
(8)	YTICK	Interval between major tick marks. (Engineering units)
(9)	HT	Height of letters for labels. (inches)

The PLOT subroutine will swap XMAX and XMIN if needed, so the order of these two arguments is not important. The same is true for the arguments YMAX and YMIN. If |XLEN| < 0.2, the program substitutes a value of 5.0 inches. If |YLEN| < 0.2, the program substitutes a value of 3.0 inches. (Thus, if the plot size is not specified, a default of 5" X 3" is used.) If either XLEN or YLEN is less than 1.0 inches, but greater than 0.2 inches, a value of 1.0 inches is substituted. Thus, the minimum axis length that can be generated is 1.0 inches. If HT = 0, the program substitutes a value of 0.15 inches.

Size and Scaling Options

The PLOT program can scale and size plots in many ways. The following descriptions tell how to use XMAX, XMIN, XLEN, and XTICK to control the X axis and plot width. Equivalent settings to control the Y axis and the plot height use the corresponding parameters YMAX, YMIN, YLEN, and YTICK.

Completely Manual Scaling of Linear Axis.

XMAX upper plot limit (not necessarily a numbered tick mark)

XMIN lower plot limit (not necessarily a numbered tick mark)

|XLEN| width of plot area if XTICK > 0 (length of axis), **or** distance between major (numbered) tick marks if XTICK < 0.

|XTICK| interval between major tick marks. If this number has only one non-zero digit, then minor tick marks are inserted also. The sign of XTICK determines whether the axis is sized by length (XTICK > 0) or by a fixed scale factor (XTICK < 0).

Independent Automatic Scaling of a Linear Axis.

In this mode, the PLOT subroutine will control everything about the axis except size. If the axis length is specified, an interval between tick marks will be found iteratively to get the best detail possible. If the scale factor is specified, then the size of the axis will be adjusted to show the entire range of the data. Auto-scaling is enabled by setting XMAX equal to XMIN. Depending on the value chosen for XMAX and XMIN, one of three different scaling methods will be used. Be sure that the same nonzero value is not used for both X and Y axes (that is, XMAX ≠ YMAX), or else the auto-scaling will be linked, rather than independent for each axis.

XMAX = XMIN = 0 *or* XMIN = XMAX ≠ YMAX

= 0 the axis will begin and end at a major tick mark. For example, if the data cover 33.2 - 36.3, the axis will probably go from 33.0 to 37.0.

> 0 the axis will begin and end at the exact min and max values. This shows the data with maximum detail, at the expense of the overall plot appearance. In the above example, the axis would go from 33.2 to 36.3.

< 0 the axis will include 0.0. In the above example, the axis would go from 0 to 35 or 40 (depending on the size of the axis).

|XLEN| width of plot area if XTICK ≥ 0 (length of axis), *or* distance between major (numbered) tick marks if XTICK < 0.

|XTICK| distance between major tick marks (same as for manual scaling) if XTICK < 0. Not used if XTICK ≥ 0.

Independent Semiautomatic Scaling of Linear Axis.

In this mode, the PLOT subroutine will select an interval between major tick marks, based on provided values of XMAX and XMIN. The upper and lower limits of the axis will be rounded to the next major (numbered) tick mark that includes the specified XMIN and XMAX values.

XMAX Maximum data value of interest

XMIN Minimum data value of interest

XTICK = 0

|XLEN| Width of plot area (axis length)

Linked Automatic Scaling of Both Linear Axes.

In this mode, an identical scale factor is selected for both axes. This is convenient in plotting X-Y trajectories without distortion. Usually the proportion of the plot is not known ahead of time, and some cropping will be performed in either the height or the width of the plot.

XMAX = XMIN = YMAX = YMIN ≠ 0

|XLEN| maximum allowable width[1] of plot area

|YLEN| maximum allowable height[1] of plot area

XTICK,YTICK both must be greater than 0

Completely Manual Scaling of Log Axis.

XMAX upper plot limit (not necessarily a numbered tick mark)

XMIN lower plot limit (not necessarily a numbered tick mark)

|XLEN| length of log axis if XTICK ≥ 0. Length of one decade if XTICK < 0.

XTICK value not used. Controls interpretation of XLEN.

[1]If XMAX and YMAX are > 0, then |XLEN| and |YLEN| are treated as dimensions of a rectangle (i.e., 8 x 10) that the plot must fit within. PLOT will swap |XLEN| and |YLEN| if better scaling can be obtained.

Automatic Scaling of Log Axis.

There are three modes for automatically scaling a log axis. Auto-scaling is enabled by setting XMAX equal to XMIN. Depending on the value chosen for XMAX and XMIN, one of three different scaling methods will be used.

XMAX = XMIN

= 0	the axis will be scaled to capture the entire range of data. For example, if the data cover the range of 7.6×10^{-2} to 1.4×10^{2}, the axis would typically go from 5×10^{-2} to 2×10^{2}.
> 0	the axis will be scaled to include the maximum values in the data, but will cover a range equal to XMAX. For the data in the above example and a XMAX value of 1000, the axis would go from 0.2 to 2×10^{2}.
< 0	the axis will be scaled to include the minimum values in the data, but will cover a range equal to |XMAX|. For the data in the above example and a XMAX value of -1000, the axis would go from 5×10^{-2} to 50.
|XLEN|	length of log axis if XTICK ≥ 0. Length of one decade if XTICK < 0.
XTICK	value not used, but sign (positive/negative) controls the interpretation of XLEN.

Text

The plotting subroutines draw text for labelling the axes and the data sets. Table 9 summarizes the ways that the display of text is determined.

Identifying Data Sets: the Symbol Key

A symbol key is shown within the plot space to identify symbols used in plotting, and thus to identify separate data sets. The key is used only when at least one data set uses an identifying symbol (NSYMB(i) ≠ 0) and when KEY(3) ≠ 0. If a symbol key is shown, its location is determined automatically unless KEY(3)=1,2,3,4,5, or 6. Names for the data sets must be provided in the array DESC, which should be declared as a character array. All character elements in an array must have the same length. Because the names of the data sets will probably not all have the same number of characters, some of the labels will be shorter than the array elements containing them. They should be padded with blanks to avoid the possibility of plotting "garbage characters" that can result from undefined variables in Fortran.

Table 9. Use of Text by the PLOT Subroutine.

Size	The size of all characters printed in the plots can be specified as HT. The value selected for HT is taken into account by all of the auto-scaling routines.
Positioning	The positioning of the title, the axes labels, and the axes numbers is done by the PLOT subroutine.
Subscripts, Superscripts	Subscripts and superscripts are entered using the ^ character to move characters up and the ~ character to move them down. $x^2 + y^2 = r^2$ would be specified as: x^2~ + y^2~ = r^2~.
Axis Numbers	Numbering of an axis can be omitted by specifying a negative XLEN.
Numerical	Formatting of numbers on linear and log scales is set by the PLOT *Formats* subroutine. Numbers are shown conventionally for $.01 \leq \|X\| < 10000$.
	Outside of that range, they are shown in scientific notation.
Long Titles	If the title provided to PLOT is too long to fit under the plot area, it will be wrapped.
Data Labels	When multiple data sets are plotted together, those plots using symbols can be identified in a Symbol Key, using labels provided to the PLOT subroutine.
Label Lengths	Lengths of the labels are determined by the program calling PLOT. The lengths are determined by the size of the character variables or strings used.
Justification	All labels passed as arguments should be left-justified and padded with blanks if necessary. Trailing blanks are ignored by PLOT. Axis labels and the title are centered; the data set labels are left-justified.

The Subroutine Call...

CALL PLOT (XYDATA, NPTS, NDSETS, NSYMB, XL, YL, TITLE, DESC, RANGE, KEY)

The PLOT subroutine has only input arguments, and it does not change any of their values when it is called.

XYDATA		real*4 array with X-Y values to be plotted. The subroutine assumes that the variables are stored as x1,y1, x2,y2, x3, ... (Thus, it is usually convenient to dimension the array as 2 × ntot, where ntot ≥ the total number of points in all of the data sets.)
NPTS	integer*4	array with number of X-Y points in each data set. (dimension to at least NDSETS)
NDSETS	integer*4	number of data sets.
NSYMB	integer*4	array with spacing information for symbols.
XL	string	X axis label.
YL	string	Y axis label.
TITLE	string	PLOT title.
DESC	string	array with labels for each data set.
RANGE	real*4	array containing 9 scaling parameters.
KEY	integer*4	array containing 5 mode-related parameters.

Example Fortran Program That Uses PLOT

The following example program is written in FORTRAN 77 and generates a plot using two data sets. The plot is shown in Figure 5. For the first data set, a symbol is shown for every 10th point and all points are connected by a solid line (NSYMB(1) = 10); for the second data set, a symbol is shown for every 2nd point, and no lines are shown connecting points (NSYMB(2) = -2). The first data set is stored in the XYDATA array in elements 1,1...2,100, and the second data set is stored in elements 1,101...2,200. The values selected for the R and K arrays result in complete auto-scaling and simple linear axes. Note that the label for the X axis includes a superscript.

```
CHARACTER*40 XL / 'Wavenumber - 1/ft'/
CHARACTER*40 YL / 'Amplitude^2~ of something'/
CHARACTER*80 TITLE / 'Example use of the PLOT subroutine'/
CHARACTER*32 DESC(2) /'Baselength = 2','Baselength = 3'/
INTEGER NPTS(2) / 100, 100/ , NSYMB(2) / 10, -2 /
INTEGER K(5) /0,1,-1,0,0/
REAL XYDATA(2,200)
REAL R(9) /0., 0.,0.,0.,5.,3.5,0.,0.,.15/
```

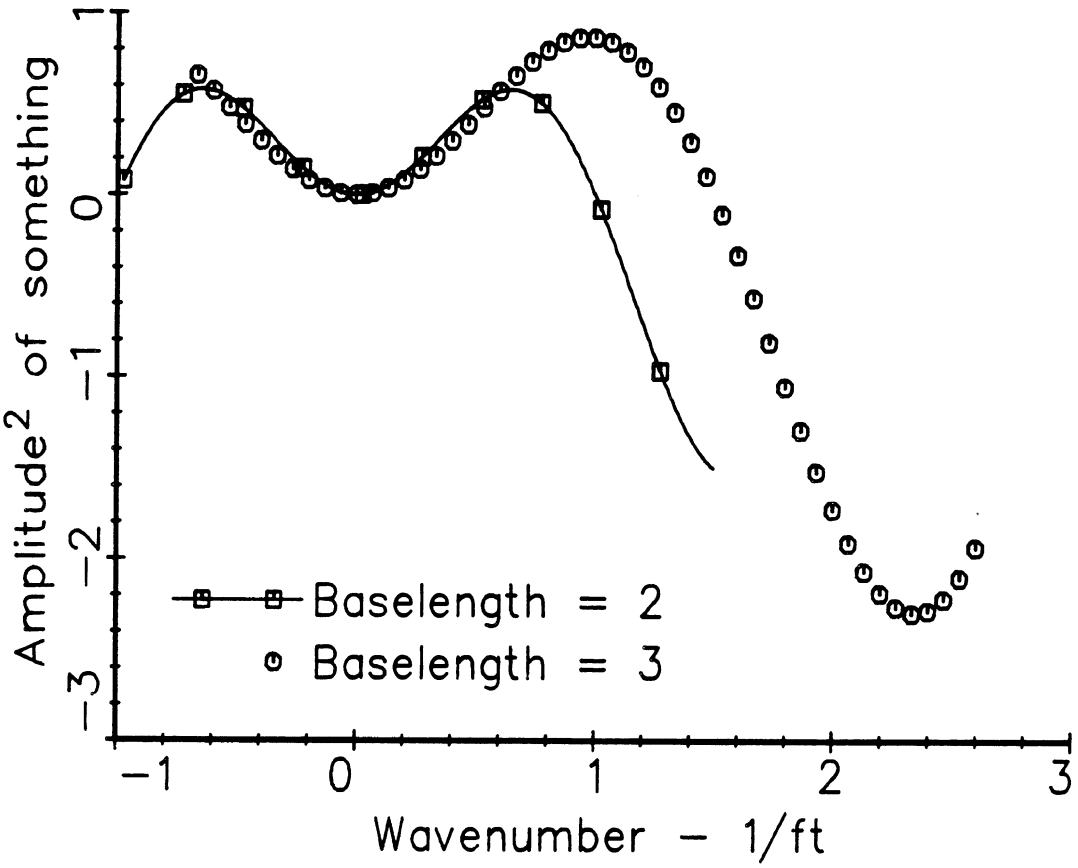


Figure 5. Example use of the PLOT subroutine

```
*
      DO 10 I = 1, 100
        X1 = -1. + I / 40.
        X2 = -0.7 + I / 30.
        XYDATA(1, I) = X1
        XYDATA(2, I) =  X1 * SIN(X1 * 6.28 / 2.)
        XYDATA(1, I + 100) = X2
        XYDATA(2, I + 100) = X2 * SIN(X2 * 6.28 / 3.)
   10 CONTINUE
*
      CALL PLOT(XYDATA, NPTS, 2, NSYMB, XL, YL, TITLE, DESC, R, K)
      END
```

6.2 The FP Program (File Plotter)

Description

The FP program reads numerical data from a text file and passes the values to the PLOT subroutine, which generates the graphics. The name of the file and the format used to store the data within the file are requested when the program runs, so that it is fairly flexible and can deal with a wide variety of file and format combinations. When specifying the file name, any legal MTS name can be used, including line number ranges within files or concatenation of multiple files. The FP program will read until reaching the end-of-file, or until reaching a line in the text file that causes a read error. If a new file is not specified for the next data set, reading will continue where the previous data set ended within the same file. Thus, a file can be subdivided by inserting a line of non-numerical characters that will cause reading to stop for one data set, and resume at the next line for the start of the next data set. This method is shown in the example below. To read only a portion of a file, it is necessary to specify the range of line numbers when entering the file name; otherwise, the entire file will be read.

The format statement should tell how the program will read a pair of X and Y values from the file, and is the part of a FORTRAN format statement lying between the parentheses. (The parentheses must also be included.) For example, the format

(10X,F10.2,T50,F10.2)

would tell the FP program to skip 10 spaces and read the X value, then read the Y value starting at position 50.

If the format statement includes a specifier for only one variable, then FP will only read Y values, and will calculate the X values based on the min and max values specified for the X axis. For example, the format (G13.6) would be interpreted by FP to mean that only the Y values are to be read.

It is not necessary that the X value precede the Y value if they both appear on the same line in the text file (the tab feature in FORTRAN can be used if the X variable appears on the line after the Y variable, e.g., [T40,F10.2,T20,F10.2]). Nor is it necessary that the X

and Y values be stored on the same lines. However, the file is always read from top to bottom, and the format statement has no provisions for rewinding. The format statement can be used to skip lines, in case a file has many points and not all are to be plotted.

Example Session

An example session follows that illustrates use of the FP program. In this example dialog, the statements that you would type at the keyboard to run the program are shown in this type face; `characters and messages from MTS are shown in this type face`, and comments are shown normally. In many cases, a default value is accepted by pressing the Return key. These responses are indicated as «Return».

```
#list fp.1
      1     2.5,200.
      2     5.,120.
      3     12.,120.
#list fp.2
      1     3.,220.
      2     6.,140.
      3     11.,190.
      4     ccccccccccccccccccc
      5     4.,90.
      6     9.,100.
      7     10.,80.
      8     15.,4.
```

The two files containing data are called fp.1 and fp.2 in this example. They are listed to show the numbers that will be plotted. Note that line 4 in the file fp.2 will cause a read error, so that only the first 3 lines will be read as data.

```
#create demo.p
 File "DEMO.P" has been created.
```

A file is created to store the plotting commands generated by the *plotsys subroutines. The file can be processed to obtain a hard-copy using the *ccqueue program, or viewed later using the *plotsee program.

```
#r svn1:pl.fp.o+svn1:pl.lib.o+*ig+*plotsys
```

The file *svn1:pl.fp.o* contains the FP program, and the file *svn1:pl.lib.o* contains the PLOT subroutine and other supporting subroutines. *IG is used so that the graphics will be shown immediately on the screen. **Plotsys* contains the software that executes many of the drawing commands.

```
#Execution begins
 Do you want extra instructions? {Y or n, def=N} «Return»
```

To save space in this listing, extra instructions are not requested. It is a good idea to answer yes if you have not used FP before.

```
How many data sets? {1 - 20, 0 ==> quit, def= 1} 3
Scaling data: {def= 0, 0, 0, 0, 5, 3, 0.5, 0.5, 0.15}  «Return»
```

The default values indicate max and min values of 0, which will force auto-scaling. The default plot size is 5" X 3", and the default height of text characters is 0.15".

```
Specify all 5 Option keys {def= 0, 1, -1, 0, 0} 3,4,-1,
```

The default setting would give a linear plot [KEY(1) = 0], simple axes with no grid [KEY(2) = 1], automatic location of the labels for the data sets [KEY(3) = -1], normal auto-scaling, and no stacking [KEY(4) = 0 = KEY(5)]. Instead, log-log scaling is specified [KEY(1) = 3], a full grid is to be drawn [KEY(2) = 4], and labels are to be automatically located. Unless all of the defaults are accepted, then all parameter values must be entered. (The last two values are both zero so entry of 0,0 at the end of the line was not needed.)

```
Enter one skip value (N) for each data set.
{Def=  0, 0, 0}  -20000,1,1,
```

The default of 0,0,0, means that each data set would be represented by a line plot, with no identifying symbols. Instead, the first data set will be shown as a thick line plot (N < -10000, see Subsection 6.1). The second and third data sets will be shown with line plots, with an identifying symbol at each point.

```
Enter Title for X axis {def=time - sec}
Frequency - Hz
Enter Title for Y axis {def=amplitude}
«Return»
Enter Title for Plot {def below:

Demo plot made with the FP program
Label for Data Set # 1 {def=Data Set #1}  Boundary
File Name: {def=*SOURCE*}  fp.1
X-Y FORMAT: {def=(2F10.2)}  «Return»

Label for Data Set # 2 {def=Data Set #2}  «Return»
File Name: {def=fp.1}  fp.2
X-Y FORMAT: {def=(2F10.2)}  «Return»

Label for Data Set # 3 {def=Data Set #3}  experimental data
File Name: {def=fp.2}  «Return»
X-Y FORMAT: {def=(2F10.2)}  «Return»
```

The first data set is read from the file fp.1, while the second and third sets are read from file fp.2. All three are read using the default format of (2F10.2). Labels are specified for the first and third sets, while the default of "Data Set #2" was accepted for the second. On a graphics terminal, such as a Mac running Versaterm, the plot shown in Figure 6 is drawn on the screen. The session continues...

```
 Blow-up, Redraw, Plot or Continue? p
 9        was referenced, but unit is not set.
Enter a new file/device name, "CANCEL", or "HELP".
?demo.p
```

To obtain a Calcomp hard-copy of the plot, it is necessary to store the pen instructions in a file attached to unit 9. This is indicated by the answer P. Since unit 9 was not specified at run time, MTS prompts for a file name, and we answer with the name of the file created earlier.

```
 PDS: PLOT DESCRIPTION GENERATION BEGINS.
 Blow-up, Redraw, Plot or Continue? c
```

Continue by returning from the plot interface back to the FP program.

```
 How many data sets? (1 - 20, 0 ==> quit, def= 3) 0
```

Quit the FP program by entering 0.

```
#Execution terminated
```

6.3 The FUNCP Program (Function Plotter)

The FUNCP program is used to plot mathematically defined functions. To use this program, write a FORTRAN function of the form FUNCTION F (X, I) where *F* is the function of *X* and *I* is the function number, needed when showing several functions in the same plot. For example, to plot the two functions

$$f_1 = 3 + 4 \times X$$

$$f_2 = 2 + 3 \times X - .2 \times X^2$$

we write the FORTRAN function

```
      FUNCTION F (X, I)
      IF (I .EQ. 1) F = 3. + 4. * X
      IF (I .EQ. 2) F = 2. + 3 * X - 0.2 * X * X
      IF (I .GT. 2) F = 0
      RETURN
      END
```

After compiling the FORTRAN function, run the plotter by typing: *sou svn1:funcp*. MTS will respond by saying that it cannot find the function F, and will request the name of the file containing it. Enter the name of the file containing the compiled function, and continue. The FUNCP program is similar to FP in operation, except that all of the data are generated

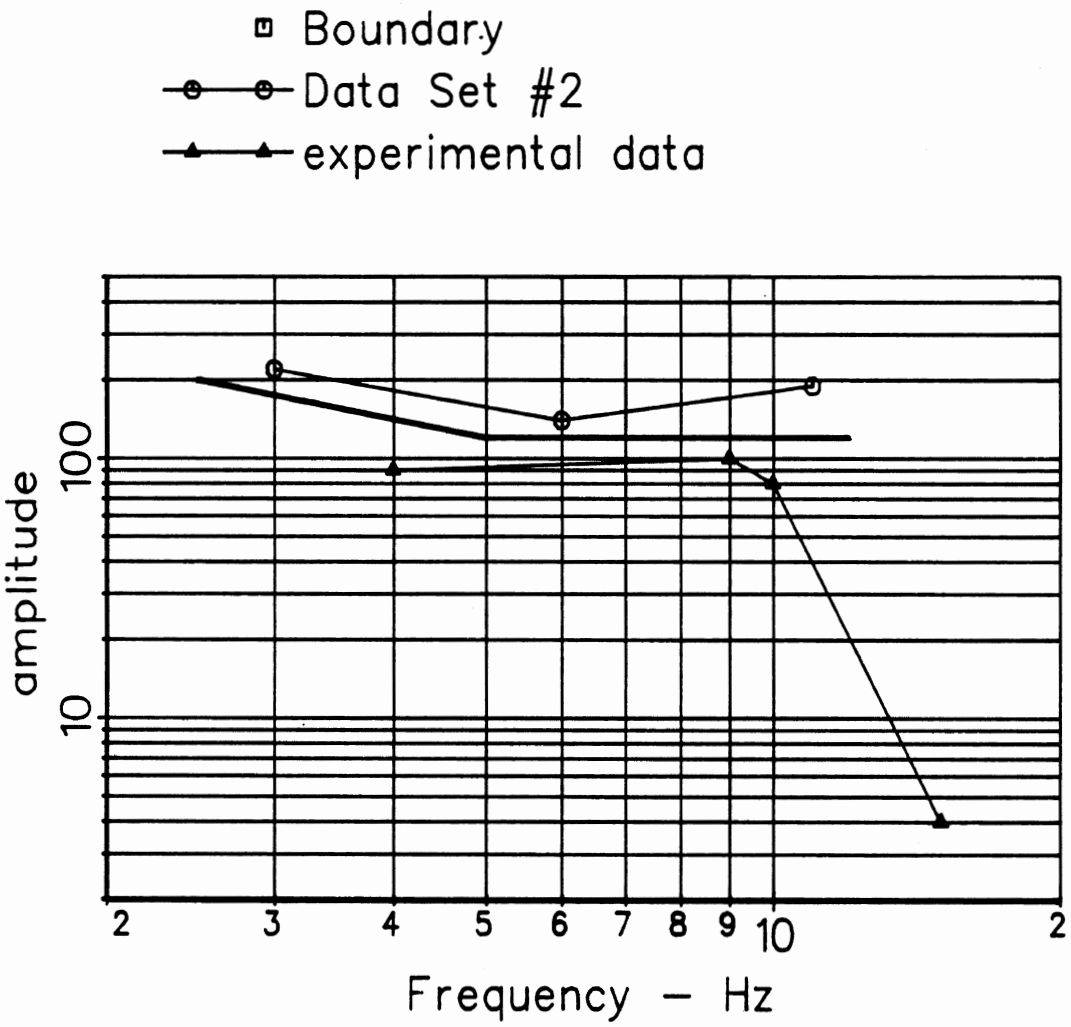


Figure 6. Demo plot made with the FP program

using the provided function. The range of X values is determined by the plotting range. Constant intervals are used for linear scaling, and constant ratios are used for log scaling.

6.4 Plotter Support Subroutines

The subroutines in this library were all written in FORTRAN 77. All real*4 arguments will either have units of inches, corresponding to the size of something plotted on paper, or else engineering units, corresponding to whatever is being plotted. The subroutines are all listed in Table 10 and described below. In the descriptions, the symbols →, ←, and ↔ are used to indicate that subroutine arguments are input only, output only, or updated by the subroutine. Each subprogram is a subroutine, unless it is specifically identified as a function. Arguments with type "string" are character variables. The length of a string argument is inherited from the calling program. The source code for these routines is contained in the MTS file *svn1:pl.subs.s*; the compiled code is contained in the library file *snv1:pl.lib.o*.

AUTOLG (DMAX, DMIN, MAX, MIN)

Choose (optionally) max and min values for a log axis.

→ DMAX	real*4	maximum value of data (engineering units).
→ DMIN	real*4	minimum value of data (engineering units).
↔ MAX,		
↔ MIN	real*4	max, min values to be used on the axis (engineering units). If XMIN ≠ XMAX, then do not do anything. If XMIN = XMAX then: = 0 → set max, min to include entire range of data. > 0 → set max to include DMAX. Set min so that the ratio max/min is the original XMAX. < 0 → set min to include DMIN. Set max so that the ratio max/min is the original |XMAX|.

AUTOLN (DMAX, DMIN, MAX, MIN, TICK, AXLEN, HT)

Choose tick interval (and optionally max and min values) for a linear axis.

→ DMAX	real*4	maximum value of data (engineering units).
→ DMIN	real*4	minimum value of data (engineering units).
↔ MAX,		
↔ MIN	real*4	max, min values to be used on the axis (engineering units). If XMIN = XMAX then subroutine will select new values. If XMIN = XMAX < 0 then axis will include 0.
← TICK	real*4	interval between major tick marks (engineering units).

Table 10. Plotting Subroutines.

AUTOLG (X1, X2, XMAX, XMIN) — select max and min values for a log axis.
AUTOLN (DMAX, DMIN, XMAX, XMIN, TICK, XLEN, HT) — select max, min, and tick interval for a linear axis.
AUTOTK (AXLEN, MAX, MIN, TICK, HT) — calculate tick interval for a linear axis.
Function DLENST (HT, STRING) — length of a string of text when it is drawn.
DRWSTR (X, Y, HT, STRING, ANGLE) — draw a string of text.
ENGR (X, Y, X0, Y0, DX, DY, XMIN, XMAX, LOGLIN) — transform coordinates.
LABEL (X, STRING, L) — convert number to string to label tick mark on axis.
LINAX (X, Y, AXLEN, HT, ANGLE, MIN, MAX, TICK, NTICK, TITLE) — draw linear axis.
LINGRD (X, Y, AXLEN1, AXLEN2, ANGLE, MIN, MAX, TICK) — draw 1/2 linear grid.
LOGAX (X, Y, AXLEN, HT, ANGLE, MIN, MAX, TITLE) — draw log axis.
LOGGRD (X, Y, AXLEN1, AXLEN2, ANGLE, MIN, MAX) — draw 1/2 log grid.
MAXMIN (XYDATA, XMAX, XMIN, YMAX, YMIN, N, IG) — search for max and min values in data.
PAPER (X, Y, X0, Y0, DX, DY, XMIN, XMAX, LOGLIN) — transform coordinates.
PLOT (XYDATA, NPTS, NDSETS, NSYMB, XL, YL, TITLE, DESC, RANGE, KEY) — draw complete plot using provided data and labels.
SCLDWN (X, XNORM, XDOWN) — round a number down.
SCLUP (X, XNORM, XUP) — round a number up.
TIKDRW (X, Y, COSA, SINA, SIZE) — draw a tick mark.
TIKRND (MIN, MAX, TICK) — round off min and max values based on tick interval.
TIKSET (MIN, MAX, TICK, TMIN, TMAX) — calculate first and last tick values.
X1EQX2 (X1, X2) — don't let X1 = X2.

→ AXLEN	real*4	length of the axis (inches).
→ HT	real*4	height of labels and numbers (inches).

AUTOTK (AXLEN, MAX, MIN, TICK, HT)

Choose tick interval for a linear axis.

→ AXLEN	real*4	length of the axis (inches).
↔ MAX	real*4	maximum value to be used on the axis (engineering units).
↔ MIN	real*4	minimum value to be used on the axis (engineering units).
← TICK	real*4	interval between major tick marks (engineering units).
→ HT	real*4	height of labels and numbers (inches).

function DLENST (HT, STRING)

Calculate the length of a string of text when drawn with the DRWSTR subroutine.

← DLENST	real*4	length of text when drawn (inches).
→ HT	real*4	height of letters (inches)
→ STRING	string	text to be written.

DRWSTR (X, Y, HT, STRING, ANGLE)

Draw a string of text. Subscripts and superscripts are supported through the characters: '~' = "move up" and '^' = "move down."

→ X	real*4	X coordinate where the text starts (inches).
→ Y	real*4	Y coordinate where the text starts (inches).
→ HT	real*4	height of letters (inches)
→ STRING	string	text to be written.
→ ANGLE	real*4	orientation angle of the text (degrees). 0°=horizontal (left-to-right), 90°=vertical (bottom-to-top), 180°=horizontal, upside-down...

ENGR (X, Y, X0, Y0, DX, DY, XMIN, XMAX, LOGLIN)

transform coordinates from paper (inches) to engineering units.

↔ X, Y	real*4	coordinates that get transformed.
→ X0, Y0	real*4	paper coordinates of lower-left corner of viewport (inches).
→ DX, DY	real*4	scale factors.
→ XMIN, → YMIN	real*4	engineering coordinates of lower-left corner of viewport (engineering units).

→	LOGLIN	integer	0=linear X, linear Y; 1=linear X, log Y; 2=log X, linear Y; 3=log X, log Y

LABEL (X, STRING, L)

Write a number into a string for use as a label on an axis. Fixed format is used for .1 ≤ X < 100,000. Scientific notation is used for numbers outside that range (the '~' character is included to get a superscript from the DRWSTR subroutine).

→	X	real*4	number to be converted.
←	STRING	string	string containing text version of number.
←	L	integer	number of significant characters in STRING.

LINAX (X, Y, AXLEN, HT, ANGLE, MIN, MAX, TICK, NTICK, TITLE, NCHAR)

Draw a linear axis.

→	X	real*4	dual-purpose variable: |X| is the absolute X coordinate of the axis start (inches). If X > 0, then the title is centered on the axis. If X < 0, *AND* if MAX > 0 and MIN < 0, then the label is centered on one side of zero.
→	Y	real*4	|Y| is the absolute Y coordinate of the axis start (inches).
→	AXLEN	real*4	dual-purpose variable: |AXLEN| is the length of the axis (inches). If AXLEN > 0, the major tick marks are labeled with corresponding numbers. If AXLEN < 0, the numbers are omitted.
→	HT	real*4	dual-purpose variable: |HT| = height of letters and numbers used to write labels (inches). If HT > 0, labels go on counterclockwise side of axis; if HT < 0 they go on the clockwise side.
→	ANGLE	real*4	orientation angle of the axis (degrees). 0°=horizontal (left-to-right), 90°=vertical (bottom-to-top), 180°=horizontal, upside-down...
→	MIN	real*4	starting value for axis (engineering units).
→	MAX	real*4	ending value for axis (engineering units)
→	TICK	real*4	interval between major tick marks (engineering units).
→	NTICK	integer	number of small tick marks between major ticks. For example, if TICK = 1.0 and NTICK = 5, minor tick marks would be drawn at 0.2 intervals.
→	TITLE	string	label for the axis.

The ingredients in a complete axis are: a line; major tick marks that are optionally labeled with numbers; minor tick marks that are unlabeled; and a title. The axis can be oriented in any direction, the numbering can be on either side of the axis or omitted, and the title can be centered using one of two criteria. The axis does not necessarily begin or end

on a tick mark—the ticks are located such that a tick would be located at zero. For example, if the axis covers 33.4 to 39.3, and the tick interval is specified as 2.0, then major tick marks would be drawn and optionally labelled at 34, 36, and 38.

LINGRD (X, Y, AXLEN1, AXLEN2, ANGLE, MIN, MAX, TICK)

Draw 1/2 of a grid to correspond with a linear axis drawn using LINAX.

→ X	real*4	X coordinate of the start of the grid axis (inches).
→ Y	real*4	Y coordinate of the start of the grid axis (inches).
→ AXLEN1	real*4	length of the grid axis (inches). (This should also be the length of the reference axis.)
→ AXLEN2	real*4	length of the second axis (inches).
→ ANGLE	real*4	orientation angle of the reference axis (degrees). 0°=horizontal (left-to-right), 90°=vertical (bottom-to-top), 180°=horizontal, upside-down...
→ MIN	real*4	starting value for reference axis (engineering units).
→ MAX	real*4	ending value for reference axis (engineering units).
→ TICK	real*4	interval between grid lines (engineering units).

The grid lines are perpendicular to the reference linear axis, have the same length as the second axis, and are centered on an axis that parallels the reference axis. The grid lines are drawn at the same locations as the major tick marks. The grid axis should be located exactly in the center of the plotting area, and should have the same length as the reference axis.

LOGAX (X, Y, AXLEN, HT, ANGLE, MIN, MAX, TITLE)

Draw a log axis.

→ X	real*4	|X| is the absolute X coordinate of the axis start (inches).
→ Y	real*4	|Y| is the absolute Y coordinate of the axis start (inches).
→ AXLEN	real*4	dual-purpose variable: |AXLEN| is the length of the axis (inches). If AXLEN > 0, the major tick marks are labelled with the corresponding numbers. If AXLEN < 0, the numbers are omitted.
→ HT	real*4	dual-purpose variable: |HT| = height of letters and numbers used to write labels (inches). If HT > 0, labels go on counterclockwise side of axis; if HT < 0 they go on the clockwise side.
→ ANGLE	real*4	orientation angle of the axis (degrees). 0°=horizontal (left-to-right), 90°=vertical (bottom-to-top), 180°=horizontal, upside-down...
→ MIN	real*4	starting value for axis (engineering units).
→ MAX	real*4	ending value for axis (engineering units)

→	TITLE	string	label for the axis.

The ingredients in a complete axis are: a line with major tick marks that are optionally labeled with numbers, minor tick marks that are unlabeled, and a title. Through various options, the axis can be oriented in any direction, and the numbering can be on either side of the axis or omitted. The axis does not necessarily begin or end on a tick mark.

LOGGRD (X, Y, AXLEN1, AXLEN2, ANGLE, MIN, MAX)

Draw 1/2 of a grid to correspond with a log axis drawn using LOGAX.

→	X	real*4	|X| is the X coordinate start of the grid axis (inches).
→	Y	real*4	|Y| is the Y coordinate start of the grid axis (inches).
→	AXLEN1	real*4	|AXLEN1| is the length of the grid axis (inches). (This should be the same as the length of the reference axis.)
→	AXLEN2	real*4	length of the second axis (inches).
→	ANGLE	real*4	orientation angle of the reference axis (degrees). 0°=horizontal (left-to-right), 90°=vertical (bottom-to-top), 180°=horizontal, upside-down...
→	MIN	real*4	starting value for reference axis (engineering units)
→	MAX	real*4	ending value for reference axis (engineering units)

The grid lines are perpendicular to the reference log axis, have the same length as the second axis, and are centered on an axis that parallels the reference axis. The grid lines are drawn at the same locations as the major tick marks. The grid axis should be located exactly in the center of the plotting area, and should have the same length as the reference axis.

MAXMIN (XYDATA, XMAX, XMIN, YMAX, YMIN, N, IG)

Search data for max and min values. This may use input routines to get max and min values interactively from user.

→	XYDATA	real*4	array with pairs of X and Y values. The subroutine assumes that the variables are stored as x_1, y_1, x_2, y_2, x_3, ... (The array is probably dimensioned XYDATA(2,N).)
↔	XMAX,		
↔	XMIN	real*4	dual meaning. If initially XMAX ≠ XMIN, then the subroutine will constrain the search for YMAX and YMIN to those points where XMIN ≤ X ≤ XMAX. Otherwise, MAXMIN changes XMAX and XMIN to the maximum and minimum X values found, subject to the constraints of the search.
↔	YMIN,		

↔ YMAX	real*4	dual meaning—similar to XMAX and XMIN. If YMAX ≠ YMIN, then the subroutine will constrain the search for XMAX and XMIN. Otherwise, MAXMIN changes YMIN to the minimum Y value found, subject to the constraints of the search.
→ N	integer*4	number of pairs of data points searched in XYDATA.
→ IG	integer*4	key that indicates the degree of user intervention. If IG = 0, the subroutine simply returns the max and min values that were requested. If IG < 0, the subroutine prints the max and min values that were found to unit 6. If IG > 0, the subroutine prints the max and min values that were found, and then requires the user to enter replacement values (units 5 and 6 are used).

MAXMIN has three modes for searching. If the inputs XMAX = XMIN and YMAX = YMIN, it will search the data for the absolute max and min values for both X and Y. If XMAX = XMIN, but YMAX ≠ YMIN, it will selectively search for max and min X values for only those points whose Y values are within the range defined by the limits YMIN and YMAX. Finally, if YMAX = YMIN, but XMAX ≠ XMIN, it will selectively search for max and min Y values for only those points whose X values are within the range defined by the limits XMIN and XMAX. The subroutine also has three modes for updating the max and min values, which involve optional interactive scaling.

PAPER (X, Y, X0, Y0, DX, DY, XMIN, XMAX, LOGLIN)

Transform coordinates from paper (inches) to engineering units.

↔ X, Y	real*4	coordinates that get transformed.
→ X0, Y0	real*4	paper coordinates of lower-left corner of viewport (inches).
→ DX, DY	real*4	scale factors.
→ XMIN, → YMIN	real*4	engineering coordinates of lower-left corner of viewport (engineering units).
→ LOGLIN	integer	0=linear X, linear Y; 1=linear X, log Y; 2=log X, linear Y; 3=log X, log Y

SCLDWN (X, XNORM, XDOWN)

Round off the mantissa of a variable down to the next multiple of 1, 2, or 5.

→ X	real*4	number to be rescaled (not changed by subroutine). This number must be ≥ 0.
← XNORM	real*4	normalized input, rescaled to lie between 1 and 10. For example, if X = 33.7, XNORM = 3.37.

←	XDOWN	real*4	rounded output. In the example where X = 33.7, XDOWN would be 20.

SCLUP (X, XNORM, XUP)

Round off the mantissa of a variable up to the next multiple of 1, 2, or 5.

→	X	real*4	number to be rescaled (not changed by subroutine). This number must be ≥ 0.
←	XNORM	real*4	normalized input, rescaled to lie between 1 and 10. For example, if X = 33.7, XNORM = 3.37.
←	XUP	real*4	rounded output. In the example where X = 33.7, XUP would be 50.

TIKDRW (X, Y, COSA, SINA, SIZE)

Draw a tick mark.

→	X	real*4	X coordinate of the center of the tick mark (inches).
→	Y	real*4	Y coordinate of the center of the tick mark (inches).
→	COSA	real*4	$\cos(\theta)$, where $\theta = 0$ for vertical tick and $\theta=90°$ for horizontal tick.
→	SINA	real*4	$\sin(\theta)$, where $\theta = 0$ for vertical tick and $\theta=90°$ for horizontal tick.
→	SIZE	real*4	1/2 the length of the tick line (inches).

TIKRND (MIN, MAX, TICK)

Round off max and min values using tick interval as basis of the roundoff.

↔	MIN	real*4	lower limit of axis (engineering units).
↔	MAX	real*4	upper limit of axis (engineering units).
→	TICK	real*4	tick interval (engineering units).

TIKSET (MIN, MAX, TICK, TMIN, TMAX)

Calculate the first and last major tick marks for a linear axis.

→	MIN	real*4	lower limit of axis (engineering units).
→	MAX	real*4	upper limit of axis (engineering units).
→	TICK	real*4	interval between tick marks.
←	TMIN	real*4	minimum value for tick mark within range specified by XMIN and XMAX (engineering units).

← TMAX	real*4	maximum value for tick mark within range specified by XMIN and XMAX (engineering units).

X1EQX2 (X1, X2)

Don't let X1 = X2. If they are equal, subroutine modifies both so that plot axis will have a non-zero range.

↔ X1	real*4
↔ X2	real*4

7. USER INPUT SUBROUTINES

There is a library of subroutines used to permit fairly free-form input of names and numbers into a FORTRAN 77 program, much in the same way that input is allowed in BASIC. The library is contained in the file *snv1:pl.lib.o*. Table 11 lists the subroutines, which are described in greater detail below. The source code for these subroutines is contained in the file *svn1:in.subs.s*.

7.1 Format for User Input

The subroutines get names and numbers using nearly identical formats, and this format is illustrated by the following example for getting two real numbers:

```
XLEN = 6.4
YLEN = 4.0
CALL GETX2 (-5, 6, 'Enter X and Y dimensions', XLEN, YLEN, IERR)
IF (IERR .EQ. 1) GO TO 100
```

This code would result in a screen display reading:

```
Enter X and Y dimensions {def=6.4, 4} _
```

Several responses to the example are shown in Table 12, along with the updated values of the variable XLEN and YLEN. If the user hits the *Return* key or *Ctrl-C*, the values of XLEN and YLEN are unchanged, remaining at 6.4 and 4. If *Ctrl-C* is entered, the variable IERR is set to 1, possibly indicating that the user wanted to cancel that phase of the program. (In this example, the program would branch to line 100 if a *Ctrl-C* had been entered. If the argument IR had been given a positive value of 5, then a user entry of *Ctrl-C* would cause MTS to stop the program at that point.)

When there are multiple numbers on the same line (in this example, there are two), the entries must be separated by a comma or slash "/" delimitor. Any spaces that are entered are ignored. If some but not all of the numbers are entered, then the numbers not entered will be assigned a value of zero. It is also possible to accept the default value for one variable while updating another, by using delimitors to hold the place of the variable while not actually entering a number. MTS commands will be intercepted if the line begins with a $ character. If this occurs, control resumes where it left off after the MTS command is finished.

7.2 Description of the Subroutines

Standard Subroutine Arguments

The following 4 arguments are used identically in most of the subroutines:

→ IR	integer*4	FORTRAN device number for reading. If IR > 0, then an end-of-file entry stops execution normally. If IR < 0, then abs(IR) is used and an end-of-file (Ctrl-C) does not stop execution of the program, but changes the error code, IERR.

Table 11. Subroutines that Support "User-Friendly" Input.

Generalized User Input

GETCH (IR, IW, PROMPT, CHAR, STRING, IERR) — get single character.
GETI (IR, IW, PROMPT, I, IERR) — get single integer.
GETI2 (IR, IW, PROMPT, I1, I2, IERR) — get two integers.
GETIN (IR, IW, PROMPT, IN, N, IERR) — get array of integers.
GETST (IR, IW, PROMPT, STRING, IERR) — get string.
GETX (IR, IW, PROMPT, X, IERR) — get single floating-point number.
GETX2 (IR, IW, PROMPT, X1, X2, IERR) — get two floating-point numbers.
GETXN (IR, IW, PROMPT, XN, N, IERR) — get array of floating-point numbers.
GETYN (IR, IW, PROMPT, YES, IERR) — get yes | no answer.
OPNMTS (IR, IW, IFILE, PROMPT, CANCEL, FNAME, IERR) — open MTS file.
WRTMTS (IUNIT, FILENM) — open MTS file for writing (no user interaction).

String Manipulation

ALCAPS (STRING) — convert string to ALL CAPITAL letters.
Function CAPSTR (STRING) — convert string to ALL CAPITAL letters.
COMBIN (FIRST, CONJ, SECOND, L, SUPER) — combine 3 strings.
Function LCSTR (STRING) — convert string to lower case letters.
Function LENSTR (STRING) — number of significant characters in string.
Function SEEMTS (STRING) — look for (and execute) MTS command.
STRI (I, STRING, L) — convert integer to string.
STRIN (IN, N, STRING, L) — convert series of integers to string.
STRX (X, STRING, L) — convert floating-point number to string.
STRXN (XN, N, STRING, L) — convert series of floating-point numbers to string.
STRIP1 (STRING) — strip leading blanks from string.
STRIP2 (STRING, L) — strip zeros from end of mantissa of written number.
UNIQUE (DESC, N, ICOUNT) — count number of unique labels.

Table 12. Examples of Valid Numerical Inputs.

```
Enter X and Y dimensions {def=6.4, 4} _
```

entry	*new XLEN*	*new YLEN*
	6.4	4.0
Ctrl-C	6.4	4.0
8E2, 50.	800.	50.
800	800.	0.
,0.5 E02	6.4	50.
800.000, ,	800.	4.0
5/3	5.	3.
//	6.4	4.0
800 50	80050.	0.
$Filestatus	*(MTS lists all files, then the subroutine repeats the prompt with the same default settings.)*	

→ IW	integer*4	FORTRAN device number for writing. If IW > 0, a full prompt is generated (see example below). If IW < 0 then abs(IW) is used and the first { is not printed. (This is useful when the first { was printed in a message generated by the calling program.) If IW = 0, there is no prompt at all. (The {def=...} part of the prompt can be suppressed only for string inputs, as noted in the specific subroutine descriptions.)
→ PROMPT	character	prompt string (should not include 'def='). The subroutines will get the length of this string from the calling program and print the complete length, including spaces. (The length can be shortened by specifying a substring in the calling program.)
← IERR	integer*4	Error return code. 0=ok, 1=end-of-file (Ctrl-C) was entered. Note: IERR = 1 can only occur when IR < 0.

Input Subroutines

The subroutines and their arguments are described below.

GETCH (IR, IW, PROMPT, CHAR, STRING, IERR)

Get a character interactively from the keyboard, and check the response.

↔ CHAR	character*1	character that is updated by the subroutine.
→ STRING	character	string containing a list of the acceptable answers. (The entire length of the string, as it is defined in the calling program, is used.)

When using this subroutine, it will often be best to provide the prompt message separately and to suppress the first "{" by entering a negative value of IW. For example:

```
CHARACTER*1 CHAR
CHAR = 'M'
WRITE (6, '(''&Larry, Moe, or Curly? {L, M, or C,'')' )
CALL GETCH (5, -6, ' ', CHAR, 'LlMmCc', IERR)
```

produces the screen display:

```
Larry, Moe, or Curly? {L, M, or C, def=M} _
```

Note that the list of answers includes both the upper and lower case versions of the three acceptable responses, so that there are actually six valid answers. A different answer will result in an error message that prints the acceptable answers and asks the user to try again.

GETI (IR, IW, PROMPT, I, IERR)

Get a single integer number interactively from the keyboard.

↔ I	integer*4	variable that is updated by the subroutine.

GETI2 (IR, IW, PROMPT, I1, I2, IERR)

Get two integers interactively from the keyboard.

↔ I1, I2	integer*4	variables that are updated by the subroutine.

GETIN (IR, IW, PROMPT, IN, N, IERR)

Get an array of integers interactively from the keyboard.

↔ IN	integer*4	array of variables that are updated by the subroutine.
→ N	integer*4	number of elements in the IN array.

GETST (IR, IW, PROMPT, STRING, IERR)

Get a string interactively from the keyboard.

↔ STRING	character	string variable that is updated by the subroutine. The length of this string is defined by the calling program. If this name is "NO DEFAULT" when the subroutine is called, then a default name is not printed; an attempt to accept a default name by entering a blank line results in an error message and a prompt to try again.

GETX (IR, IW, PROMPT, X, IERR)

Get a single real number interactively from the keyboard.

↔ X	real*4	variable that is updated by the subroutine.

GETX2 (IR, IW, PROMPT, X1, X2, IERR)

Get two real numbers interactively from the keyboard.

↔ X1, X2	real*4	variables that are updated by the subroutine.

GETXN (IR, IW, PROMPT, XN, N, IERR)

Get an array of real numbers interactively from the keyboard.

↔ XN	real*4	array of variables that are updated by the subroutine.
→ N	integer*4	number of elements in the XN array.

GETYN (IR, IW, PROMPT, YES, IERR)

Get a yes/no answer interactively from the keyboard.

↔ YES	logical*4	.true. if answer is yes, .false. if answer is no.

OPNMTS (IR, IW, IFILE, PROMPT, CANCEL, FNAME, IERR)

Open an MTS file and attach it to the specified device number.

→	IFILE	integer*4	FORTRAN device number for file to be opened. If IFILE < 0, then abs(IFILE) is used and file must already exist.
→	CANCEL	character	if this name is selected, no file is opened. This name should be ALL IN CAPS.
↔	FNAME	character	name of MTS file that is opened. If this name is "NO DEFAULT" when the subroutine is called, then a default name is not printed; an attempt to accept a default name by entering a blank line results in an error message and a prompt to try again.

WRTMTS (IFILE, FNAME)

Open an MTS file (for writing) with no user interaction. The subroutine checks to see if the file already exists to avoid potential errors using the FORTRAN Open statement.

→	IFILE	integer*4	FORTRAN device number for file to be opened.
↔	FNAME	character	name of MTS file that is to be opened.

String Manipulation Subroutines

Unless noted otherwise, these subroutines should be called from programs compiled under the FORTRAN 77 compiler. (Older FORTRAN 66 programs do not pass the lengths of character variables needed by most of these subroutines.)

ALCAPS (STRING)

Convert string to ALL CAPS.

↔	STRING	character	string that is modified by the subroutine.

Function CAPSTR (STRING)

Convert string to ALL CAPS.

←	CAPSTR	character	capitalized copy of input string.
→	STRING	character	input string.

Function COMBIN (FIRST, CONJ, SECOND, L, SUPER)

Combine two string expressions using a provided conjunction. Ex: combine 'Velocity ' and 'mph ' with ' - ' to yield 'Velocity - mph'

←	COMBIN	character	name of whole thing.
→	FIRST	character	string containing first name. Ex: 'Velocity '
→	CONJ	character	string containing conjugate. Ex: ' - '
→	SECOND	character	string containing second name. Ex: 'mph '

← L	integer	number of significant characters in COMBIN.
↔ SUPER	logical	if .true. then digits in SECOND are made superscripts for plotting.

Function LCSTR (STRING)

Convert string to lower case.

← LCSTR	character	lower case copy of input string.
→ STRING	character	input string.

Function LENSTR (STRING)

Count the number of significant characters in a left-justified string.

← LENSTR	integer*4	number of significant characters found in STRING.
→ STRING	character	string that is inspected.

Function SEEMTS (STRING)

Use to intercept MTS commands and execute them.

← SEEMTS	logical	.true. if string began with a $, .false. otherwise.
→ STRING	character	string which may or may not be an MTS command. If the first character is not '$', nothing happens. If the first character is '$', then the entire string is passed to MTS for interpretation.

STRI (I, STRING, L)

Convert integer to string, similar to the BASIC function STR$.

→ I	integer*4	number that is written into the string.
← STRING	character	string that will contain text version of the number.
← L	integer*4	number of significant characters in STRING.

STRIN (IN, N, STRING, L)

Convert series of integers to string, with commas.

→ IN	integer*4	array of numbers that is written into the string.
→ N	integer*4	number of numbers in the input array IN.
← STRING	character	string that will contain text version of the numbers.
← L	integer*4	number of significant characters in STRING.

STRX (X, STRING, L)

Convert real number to string, similar to the BASIC function STR$.

→ X	real*4	number that is written into the string.
← STRING	character	string that will contain text version of the number.

← L	integer*4	number of significant characters in STRING.

STRXN (XN, N, STRING, L)

Convert series of floating-point numbers to string, with commas.

→ XN	real*4	array of numbers that is written into the string.
→ N	integer*4	number of numbers in the input array XN.
← STRING	character	string that will contain text version of the numbers.
← L	integer*4	number of significant characters in STRING.

STRIP1 (STRING)

Strip blanks from beginning of string, and add blanks to end of string as needed.

↔ STRING	character	string that is modified by the subroutine.

STRIP2 (STRING, L)

Strip zeros from end of mantissa of number written in FORTRAN F or E format.

↔ STRING	character	string containing a written version of a floating-point number in E or F format. The string is modified by the subroutine.
← L	integer*4	number of characters in number after it is stripped.

UNIQUE (DESC, N, ICOUNT)

Count the number of unique labels in an array.

→ DESC	character	1-dimensional array of labels.
→ N	integer*4	number of labels.
← ICOUNT	integer*4	no. of unique labels.

8. SIGNAL PROCESSING SOFTWARE

8.1 Spectral Analysis Programs

There are presently two programs on the SVN1 account on MTS that can be used for transforming data in ERD files into the frequency domain. They are invoked with the MTS commands

sou svn1:.psd

sou svn1:.tf

The PSD program takes an ERD file as input and produces an ERD file as output. Each channel in the file is replaced with the PSD of that channel. In addition, a channel is added with the frequency values.

The TF program takes combinations of channels from the input file, as requested by the user, and calculates transfer function gains, phase angles, and coherence functions which are placed into the output file.

The transformed data in the output files can be viewed with the ERD plotter, described in Section 3.

8.2 Signal Processing Subroutines

Passing Arrays to the Subroutines

Often, the signal to be processed is stored in a 2-D array with other signals. If the array is dimensioned as ARRAY (NDIM, NSAMP), then the order of storage in the computer is as if the signals are scanned. If i is the sample number, and there are three channels X, Y, and z, the order is $x_1, y_1, z_1, x_2, y_2, z_2, x_3, \ldots$ The other option is to dimension the array as ARRAY (NSAMP, NDIM). In this case, the order of storage is the same as independent 1-D arrays with each signal in its own array. The storage order for the above example would be $x_1, x_2, \ldots x_{nsamp}, y_1, y_2, \cdots y_{nsamp}, z_1, \cdots z_{nsamp}$.

Knowledge of the layout is useful in calling subroutines that process signals. If the first layout is used, the subroutine must be given the first dimension NDIM, but it need not know exactly which channel is what. If the second layout is used, the subroutine can treat the data as if they were in a 1-D array. To illustrate this, several examples are given for a case where the data are in a two dimensional array dimensioned to hold three variables, and one of those variables is to be differentiated.

Case 1. The array is dimensioned ARRAY (100, 3) and we wish to differentiate the 2nd channel. The subroutine is told that the first dimension is 1 and that the array begins at the start of the second channel. The call is:

CALL DERIV (ARRAY (1, 2), 1, NSAMP, STEP)

Case 2. The array is dimensioned ARRAY (100, 3) and we wish to differentiate the middle of the third channel, from sample numbers 20 to 80 (61 points). The

subroutine is told that the first dimension is 1, and that the array begins in the middle of the second channel. The subroutine call is:

CALL DERIV (ARRAY (20, 3), 1, 61, STEP)

Case 3. The array is dimensioned ARRAY (3, 100) and we wish to differentiate the second channel. The subroutine must be told that the first dimension is 3, but we tell it that the first element is at the beginning of the second channel. The call is:

CALL DERIV (ARRAY (2, 1), 3, NSAMP, STEP)

Case 4. The array is dimensioned ARRAY (3, 100) and we wish to differentiate the middle of the third channel, from sample numbers 20 to 80 (61 points). The subroutine is told that the first dimension is 3, and that the array begins in the middle of the second channel. The subroutine call is:

CALL DERIV (ARRAY (3, 20), 3, 61, STEP)

Descriptions of the Subroutines

The subroutines that process signals are available in the library *snv1:sp.lib.o* and are listed in Table 13. The source code for these routines is in files beginning with *SVN1:sp.* and ending with *.s*. For example, the file *svn1:sp.psd.s* contains the PSD subroutine. Each subprogram is a subroutine unless it is specified below as a function. The arguments are designated as inputs (→), outputs (←), or both (↔).

DERIV (ARRAY, NDIM, NSAMP, STEP)

Differentiate a signal. The differentiation is performed by taking differences between adjacent values and dividing by STEP.

↔ ARRAY	real*4	2-D array with the signal to be processed in channel 1.
→ NDIM	integer*4	first dimension of ARRAY.
↔ NSAMP	integer*4	number of samples in ARRAY. The subroutine decreases this value by 1, because one point is lost in the processing.
→ STEP	real*4	sample interval.

Function FSPLIT (F1, F2, IBAND)

Interpolate between two frequencies.

← FSPLIT	real*4	frequency 1/2 way between F1 and F2.
→ F1, F2	real*4	input frequencies.
→ IBAND	integer*4	switch: 0 = linear split; anything else = log split.

Table 13. Signal-Processing Subroutines.

DERIV (ARRAY, NDIM, NSAMP, STEP) — differentiate signal.
Function FSPLIT (F1, F2, IBAND) — interpolate between 2 frequencies.
MSFFT (ARRAY, NSAMP, NFFT, DT, DF, DETREN, WINDOW, APLTYP) — replace signal with Fourier transform.
MSPSD (FFTSIG, FREQ, PSD, DF, NFFT, NPSD, I1PSD, NDFREQ, NDPSD, IBAND) — compute PSD function from Fourier transform.
MSTF (FFTY, FFTX, FREQ, TFG, TFP, COH, DF, NFFT, NTF, I1TF, NDFREQ, NDTFG, NDTFP, NDCOH, IBAND) — compute transfer function from 2 Fourier transforms.
HILOF (ARRAY, NDIM, NSAMP, MOVAV1, MOVAV2, IFILT) — filter signal using moving average.
LRSLOP (ARRAY, NDIM, NSAMP, SLOPE) — calculate slope using linear regression.
QCFILT (ARRAY , NDIM, NSAMP, SPEED, STEP) — filter signal using "golden" quarter-car.

HILOF (ARRAY, NDIM, NSAMP, MOVAV1, MOVAV2, IFILT)

Filter signal using moving average as either a high-pass or low-pass.

↔ ARRAY	real*4	2-D array with the signal to be processed in channel 1. This array must be dimensioned to include NSAMP + 2 × MOVAV2 points, because the subroutine needs additional space. The array elements from NSAMP + 1 to NSAMP + 2 × MOVAV2 are trashed by the routine.
→ NDIM	integer*4	first dimension of ARRAY.
↔ NSAMP	integer*4	number of samples in ARRAY. The subroutine decreases this value by 1, because one point is lost in the processing.
→ MOVAV1	integer*4	number of samples included in the moving average.
→ MOVAV2	integer*4	number of samples to center of moving average—typically set to MOVAV1/2.
→ IFILT	integer*4	switch: 1 = high-pass; 2 = low pass.

A moving average normally loses samples at the beginning and the end. This subroutine creates artificial data points to produce a filtered signal with the same number of points as the original signal. It does this by using the LRSLOP subroutine to create MOVAV2 additional points at the beginning and MOVAV2 points at the end of the signal.

LRSLOP(ARRAY, NDIM, NSAMP, SLOPE)

Calculate overall slope of a signal using linear regression.

→ ARRAY	real*4	2-D array with the signal to be processed in channel 1.
→ NDIM	integer*4	first dimension of ARRAY.
→ NSAMP	integer*4	number of samples in ARRAY.
← SLOPE	real*4	slope of the signal, with respect to sample number.

This subroutine regresses the sampled values in ARRAY against sample number. The resulting SLOPE value is useful for removing trends, or for generating artificial points at the beginning or end of a signal.

MSFFT (ARRAY, NSAMP, NFFT, DT, DF, DETREN, WINDOW, APLTYP)

Convert signal into the frequency domain.

↔ ARRAY	real*4 AND complex	As an input, this contains NIN consecutive real*4 data values. As an output, it contains NOUT complex Fourier coefficients.
→ NSAMP	integer*4	number of samples in the input signal.

← NFFT	integer*4	number of complex Fourier coefficients. NFFT will be the highest power of 2 that is not greater than NSAMP. (E.g., if NSAMP is 1500, NFFT would be 1024.)
→ DT	real*4	time interval between input points.
← DF	real*4	frequency interval between output points.
→ DETREN	integer*4	switch for detrending: 0 = off, anything else = on.
→ WINDOW		integer*4 windowing options. 0 = no window (boxcar). (No other options as of 5-86.)
→ APLTYP	integer*4	type of application for transformed data. 0 = random signals (PSD, transfer function) 1 = sinusoidal amplitude (uniformity). This determines the scale factor used on the output.

This subroutine transforms sampled data in a one-dimensional array into the frequency domain, using two FFT subroutines from the *naas:nal* library. The subroutine can be used to replace one channel in a multi-dimensional array if it is dimensioned as ARRAY (NMAX, NCHAN) where NMAX is not less than NFFT*2. The subroutine will add zeros to the end of the data as needed to obtain a power of two for the FFT. It corrects the amplitudes of the Fourier coefficients to get the same amplitudes as would have been obtained if the signal had continued. If NSAMP is not a power of two, the compensation used will depend on the value of APLTYP. When APLTYP is 0, the scaling is set so that integrals over frequency will give mean-square values. When set to 1, the peak values of spectral peaks will equal the amplitude of the sinusoid. When no zeros are added, the scaling is the same for both options.

Trend removal should be used for signals that include drift, or when signals are random and the number of points is not too large. For very long signals (more than 5000 points), the trend removal can introduce error due to roundoff.

The transformed signal will usually be processed by another subroutine to get a standard "final" form, such as a PSD or transfer function.

MSPSD (FFTSIG, FREQ, PSD, DF, NFFT, NPSD, I1PSD, NDFREQ, NDPSD, IBAND)

Compute a PSD function from the Fourier coefficients obtained with MSFFT.

→ FFTSIG	complex or real	complex Fourier coefficients. This array would normally be obtained as the output of the subroutine MSFFT. The first complex coefficient is the static component of the original signal, and is ignored by the subroutine.
→ FREQ	real*4	2-dim array with center frequencies that define averaging intervals used for PSD calculation. (Frequencies are assumed to be in channel 1.) This array should have at least NPSD + 1 values, because the subroutine uses the

		next higher center frequency to determine the cutoff frequency between bands.
← PSD	real*4	2-dim array with the calculated values of the PSD function for the frequencies contained in the FREQ array. (The data are put into channel 1.)
→ DF	real*4	frequency interval between the Fourier coefficients contained in the FFTSIG array.
→ NFFT	integer*4	no. of complex coefficients in the FFTSIG array. Since this applies to complex numbers, there are 2 * NFFT floating point numbers involved.
↔ NPSD	integer*4	as input, this is the number of points in the FREQ array. If there are not enough coefficients in the FFTSIG array to calculate the PSD values for the highest center frequency, then NPSD will be set to the highest element in the PSD array containing a valid PSD value.
← I1PSD	integer*4	first element in the PSD array containing a valid PSD value. (If DF is too coarse then the first few frequencies in the FREQ array might be too low for calculation of PSD values.) The number of PSD values is therefore NPSD - I1PSD + 1.
→ NDFREQ	integer*4	first dimension of FREQ array.
→ NDPSD	integer*4	first dimension of PSD array.
→ IBAND	integer*4	switch for options in dividing frequencies into adjacent bands. The dividing frequencies will always be 1/2 way between center frequencies. 0 = linear split, anything else = log split.

Note that the FREQ and PSD arrays can be different channels in the same array in the calling program.

MSTF (FFTY, FFTX, FREQ, TFG, TFP, COH, DF, NFFT, NTF, I1TF, NDFREQ, NDTFG, NDTFP, NDCOH, IBAND)

Compute transfer functions and the coherence function between two signals using the Fourier coefficients obtained with the MSFFT subroutine.

→ FFTY, FFTX	complex or real	complex Fourier coefficients for 2 signals. These arrays would normally be obtained as the output of the subroutine MSFFT. The first complex coefficient in each array is the static component of the original signal, and is ignored by the subroutine. FFTX contains the coefficients for the input, FFTY contains the coefficients for the output signal.
→ FREQ	real*4	2-dim array with center frequencies that define averaging intervals used for COH calculation. (Frequencies are assumed to be in channel 1.) This array should have at

		least NCOH + 1 values, because the subroutine uses the next higher center frequency to determine the cutoff frequency between bands.
← TFG	real*4	2-dim array with the calculated values of the transfer function gain relating the two signals, for the frequencies contained in the FREQ array. (The gains are put into channel 1 of TFG.) This output is suppressed when NDTFG < 1.
← TFP	real*4	Same as TFG, except this contains the phase angle (degrees) between the two signals. This output is suppressed when NDTFP < 1.
← COH	real*4	Same as TFG, except this contains the coherence between the two signals. This output is suppressed when NDCOH < 1.
→ DF	real*4	frequency interval between the Fourier coefficients contained in the FFTX and FFTY arrays.
→ NFFT	integer*4	no. of complex coefficients in the FFTX and FFTY arrays. Since this applies to complex numbers, there are 2 * NFFT floating point numbers involved.
↔ NTF	integer*4	As input, this is the number of points in the FREQ array. If there are not enough coefficients in the FFTX and FFTY arrays to calculate the output values for the highest center frequency, then NTF will be set to the highest element in the output arrays containing a valid value.
← I1TF	integer*4	first element in an output array containing a valid value. (If DF is too coarse then the first few frequencies in the FREQ array might be too low for calculation of output values.) The number of output values is therefore NTF - I1TF + 1.
→ NDFREQ	integer*4	first dimension of FREQ array.
→ NDTFG	integer*4	first dimension of TFG array. If ≤ 0, gains are not calculated.
→ NDTFP	integer*4	first dimension of TFP array. If ≤ 0, phase angles are not calculated.
→ NDCOH	integer*4	first dimension of COH array. If ≤ 0, coherence values are not calculated.
→ IBAND	integer*4	switch for options in dividing frequencies into adjacent bands. The dividing frequencies will always be 1/2 way between center frequencies. 0 = linear split, anything else = log split.

Depending on the input values used, this subroutine will provide any combination of transfer function gain, phase, and coherence. Averaging is controlled by an input array containing the center frequencies—the subroutine will average as necessary to provide the output values corresponding to frequency bands having the specified centers. It is possible to specify frequency bands that are too narrow to perform the averaging needed for valid

transfer functions. It is recommended that the separation of the frequencies in the FREQ array should be at least 5 × DF.

QCFILT (ARRAY, NDIM, NSAMP, SPEED, STEP)

Filter a signal using a quarter-car simulation.

↔ ARRAY	real*4	2-D array with the signal to be processed in channel 1. As an input, this is an elevation profile. As an output, it is a slope profile, with units of 'somethings'/m, where 'somethings' are the units of the elevation input.
→ NDIM	integer*4	first dimension of ARRAY.
↔ NSAMP	integer*4	number of samples in ARRAY. The subroutine decreases this value by 1, because one point is lost in the processing.
→ SPEED	real*4	simulation speed, with units: km/h
→ STEP	real*4	sample interval, with units of meters.

Note: this subroutine uses the matrix inversion program MINV, contained in the *naas:ssp* library.

9. OUTPUTS FROM THE VEHICLE SIMULATIONS

The *Phase-4* and *Yaw-Roll* vehicle simulation models produce ERD files containing the time histories of the simulation variables. The files can be rather large, containing over 500 channels in some cases. Thus, it is necessary to use a rational convention for identifying and locating the channels. This section describes the labels that are used. When possible, the same names are recommended for experimental data and other models to facilitate comparisons of a variable as obtained from various sources.

9.1 Labelling Conventions

The ERD files from the simulations contain labels for each channel, identified by the keywords SHORTNAM, LONGNAM, UNITSNAM, GENNAME and RIGIBODY as described in Table 4 in Section 2. In addition, several keywords are used that may help describe the run in general. In the case of the Phase-4 model, there will be from 58 to 510 data channels (where each channel corresponds to a simulation variable). The actual number depends on the number of sprung masses and axles in the simulated vehicle. For the Yaw-Roll model, there will usually be from 20 to 200 channels. Most of the time, the short names will be of primary interest, because they will be used to select channels in post-processing programs such as the plotter.

Compatibility between the Phase-4 and Yaw-Roll Models

The Phase-4 and Yaw-Roll models use different internal schemes to describe the vehicle being simulated. Also, there are certain vehicle configurations that can be simulated with one model but not the other. For example, the Phase-4 model can simulate triples combinations, while the Yaw-Roll model can only handle doubles. As another example, the Yaw-Roll model can handle arbitrary axle layouts, while the Phase-4 program is limited to a maximum of two axles on a trailer.

Naming and labelling conventions are generally based on the Yaw-Roll conventions. Axles are identified by ascending number, starting with the front axle of the tractor (axle 1) and ending with the trailing axle of the rearward trailer. Dollies are treated as sprung masses. Therefore, a doubles combination will have four sprung masses (tractor, first semitrailer, dolly, second semitrailer). A triples combination will have six sprung masses (tractor, first semitrailer, first dolly, second semitrailer, second dolly, third semitrailer).

Rigid Body Names

The RIGIBODY keyword is used to associate the variables in the file with the various rigid bodies that comprise the vehicle model. There are five classes of names, each with a different naming convention, as shown in Table 14.

Table 14. Rigid-Body Names Used in the Vehicle Simulations.

---

1. Sprung Masses: Variables associated directly with a sprung mass (e.g., roll angle, lateral acceleration) will have the RIGIBODY name for that sprung mass. At most, a simulation can have six sprung masses, and hence six RIGIBODY names. At the least, it will have one sprung mass. The names used are:

Sprung mass #1	(only one sprung mass)	*Sprung Mass*
	(combination vehicle)	*Tractor*
Sprung mass #2	(2 sprung masses)	*Semi-trailer*
	(4 or more sprung masses)	*1st Semi-trailer*
Sprung mass #3	(4 sprung masses)	*Dolly*
	(6 sprung masses)	*1st Dolly*
Sprung mass #4		*2nd Semi-trailer*
Sprung mass #5		*2nd Dolly*
Sprung mass #6		*3d Semi-trailer*

(This naming method does not recognize the B-train configuration which can be simulated with the Yaw/roll model.)

2. Axles: Variables that describe the state of an axle (X position, Y position, roll) have a RIGIBODY name for that axle, which is simply *Axle #n.* (Axle #1, Axle #2, ... Axle #13)

3. Half-Axles: Most of the variables in the ERD file are associated with one side of an axle. These variables have RIGIBODY names of either *Left side, Axle #n* or *Right side, Axle #n.*

4. Hitches: Articulation angles and hitch forces are associated with a hitch. The names used are *Hitch 1-2, Hitch 2-3, Hitch 3-4, Hitch 4-5,* and *Hitch 5-6.* For the first hitch, there are two variations:

First hitch	(only one hitch)	*Hitch*
	(more than one hitch)	*Hitch 1-2*

5. Input: Steer and braking input variables have the RIGIBODY name: *Input.*

General Names

The GENNAME keyword is used to assign a general name to each variable in the file, to aid in labelling plots of several channels. The general name for a simulation variable is the name that would be used when reference to the associated rigid body is omitted. Examples are *Lateral Acceleration, Slip Angle,* and *Y Position.* Table 15 lists all General Names currently used in the Phase-4 and Yaw-Roll simulations.

Units

The UNITSNAM keyword is used to specify the units associated with each channel, using the names shown in Table 16. The following subsections show examples of most of the unit names used.

Long Names

Each channel in the file is also given a unique name that can be up to 32 characters long and specified using the LONGNAME keyword. The long names are usually obtained by combining the general name (GENNAME) and the rigid body name (RIGIBODY) for each channel, e.g., *X Position, 1st Semi-trailer*; *X Position, Axle 3*; *Articulation Angle, Hitch 1-2*. For the half-axles, the side in the RIGIBODY name is always abbreviated as L or R, e.g., *Brake Force, L Side, Axle 10.* The word "Axle" is abbreviated as "Ax" when necessary to stay within the 32 character length limitation of a long name. Also, the GENNAME will be abbreviated as necessary, e.g., *Longitudinal Slip, R Side, Ax 12, Long. Accel., 2nd Semi-trailer.* The two input variable names are *Input Steer Angle* and *Brake Treadle Pressure.* The following subsections show examples of most of the long names used.

Short Names

Each channel in the ERD file is given a unique short name that is limited to eight characters in length. These names are specified in the file with the SHORTNAM keyword. The short names are basically abbreviated versions of the long names. As with the long names, the naming convention is determined by which of the five RIGIBODY categories that the variable falls within. Unless the name of a variable is very short (such as Yaw or Roll), the symbol for a variable is used instead of its name.

1. *Sprung Masses:* Variables associated directly with a sprung mass will have a short name that ends with two characters, #n (n is the sprung mass number), such as *X cg #3* or *Ay cg #1.* This leaves 6 characters for the general variable portion of the name.

2. *Axles:* Names of variables that describe the state of an axle will always end with the number of the axle, without the # symbol. When the number has one digit, it is always preceded by a blank space. When it has two digits, there will be a preceding blank if it will fit within the allowable 8 character length. Therefore, a maximum of 6 characters are left to describe the variable itself. The names for the

Table 15. General Variable Names Used with the Vehicle Simulations.

Units of Variable	*Names That are Used in the Simulations*		
Length	Spring Deflection Z Position	X Position	Y Position
Length-1	Curvature		
Linear Velocity	X Velocity	Y Velocity	Z Velocity
Linear Acceleration	Lateral Acceleration	Longitudinal Acceleration	
Angle	Articulation Angle Slip Angle	Pitch Angle Steer Angle	Roll Angle Yaw Angle
Angular Velocity	Pitch Rate Spin Velocity	Roll Rate	Yaw Rate
Angular Acceleration	Spin Acceleration		
Force	Brake Force Spring Force	Load	Side Force
Moment	Aligning Moment Yaw Moment	Roll Moment	Brake Torque
Pressure	Brake Pressure		
Dimensionless	Adhesion Utilization	Longitudinal Slip	

Table 16. Names of Units Used in the Vehicle Simulations.

Units	*name*
Length	ft
Exceptions: leaf-spring deflection is in, curvature is 1/ft	
Velocity	ft/sec
Acceleration	g's
Angle	deg
Angular Rate	deg/sec
Exception: spin rate of wheels is rad/sec	
Angular Acceleration	rad/s**2
(Only used for wheel spin)	
Force, Load	klbs
Moment, Torque	ft-lbs
Pressure	psi
Dimensionless	--
(i.e., long. slip, adhesion utilization)	

position variables (X, Y, Z, and Phi) will include either the word Axle or the shortened code Ax. (E.g., *X Axle 3, Phi Ax 2, Phi Ax11, Z Axle13*)

3. *Half-Axles:* These names always begin with either an L or an R, followed by a blank, to indicate the side of the axle. They also require up to 2 characters at the end of the name for the axle number, leaving a maximum of 4 characters to identify the variable. When the number has one digit, it is always preceded by a blank space. When it has two digits, there will be a preceding blank if it will fit within the allowable 8 character length. (E.g., *L Fz 2, R Fz 2, R Alph 5, R Alph11.*)

4. *Hitches:* These names will always end with 3 characters that indicate which sprung masses are connected by the hitch, e.g., *Art 1-2*. If there is room, there will also be a blank preceding the numbers.

5. *Inputs:* There are just two input variables, named *Steer in* and *Brake in.*

Additional Keywords

In addition to labels assigned to the channels, the following optional keywords are included in the ERD files generated by the simulation programs: AXLETRAK, AXLEWT, FSTAXLES, HISTORY, HITCHKEY, NAXLES, ROLLCNTR, ROLLHT, SPEEDMPH, SPRUNGWT, TRUCKSIM, XLABEL, and XUNITS. These types of header data are all defined in table 4 from Section 2.

9.2 Channel Names from the Yaw-Roll Model

The following list indicates the spelling used for the short names, long names, and unit names for most of the channels. The channel numbers are for a specific vehicle configuration (this was a five-axle doubles combination) and will not be the same for other configurations. (The list was obtained with the ERDP plotting program. The short name is given first, followed by the long name and units in the curly brackets.)

Input

```
1  Steer in  {Input steer angle - deg                    }
```

Hitch Variables

```
2  Fh1 1-2   {Load, Hitch 1-2 - klbs                     }
3  Art 1-2   {Articulation Angle, Hitch 1-2 - deg        }
4  Mx 1-2    {Roll Moment, Hitch 1-2 - ft-lbs            }
5  Mz 1-2    {Yaw Moment, Hitch 1-2 - ft-lbs             }
6  Fy 1-2    {Side Force, Hitch 1-2 - klbs               }
7  Fh1 2-3   {Load, Hitch 2-3 - klbs                     }
8  Art 2-3   {Articulation Angle, Hitch 2-3 - deg        }
9  Mx 2-3    {Roll Moment, Hitch 2-3 - ft-lbs            }
```

```
10  Mz 2-3     {Yaw Moment, Hitch 2-3 - ft-lbs                }
11  Fy 2-3     {Side Force, Hitch 2-3 - klbs                  }
12  Fh1 3-4    {Load, Hitch 3-4 - klbs                        }
13  Art 3-4    {Articulation Angle, Hitch 3-4 - deg           }
14  Mx 3-4     {Roll Moment, Hitch 3-4 - ft-lbs               }
15  Mz 3-4     {Yaw Moment, Hitch 3-4 - ft-lbs                }
16  Fy 3-4     {Side Force, Hitch 3-4 - klbs                  }
```

Sprung Mass Variables

```
17  X cg #1    {X Position, cg, Tractor - ft                  }
18  Y cg #1    {Y Position, cg, Tractor - ft                  }
19  Z cg #1    {Z Position, cg, Tractor - ft                  }
20  Roll #1    {Roll angle,  Tractor - deg                    }
21  Yaw #1     {Yaw angle,  Tractor - deg                     }
22  Pitch #1   {Pitch angle,  Tractor - deg                   }
23  v cg #1    {Y velocity, cg, Tractor - ft/sec              }
24  p of #1    {Roll rate,  Tractor - deg/sec                 }
25  r of #1    {Yaw rate,  Tractor - deg/sec                  }
26  q of #1    {Pitch rate,  Tractor - deg/sec                }
27  Ay cg #1   {Lat. accel., Tractor - g's                    }
28  Slip #1    {Slip Angle, Tractor - deg                     }
29  Rho cg#1   {Curvature, Tractor - 1/ft                     }

30  X cg #2    {X Position, cg, 1st Semi-trailer - ft         }
 •
 •
 •
```

Axle Variables

```
61  phi Ax 1   {Roll, Axle 1 - deg                            }
62  Z Axle 1   {Bounce, Axle 1 - ft                           }
63  B-str 1    {Axle Steer angle, Axle 1 - deg                }
64  X Axle 1   {X Position, Axle 1 - ft                       }
65  Y Axle 1   {Y Position, Axle 1 - ft                       }
```

Half-Axle Variables

```
66  L alph 1   {Slip angle, L side, Axle 1 - deg              }
67  L Fz 1     {Load, L side, Axle 1 - klbs                   }
68  L Fy 1     {Side force, L side, Axle 1 - klbs             }
69  L Mz 1     {Aligning moment, L side, Axle 1 - ft-lbs      }
70  L Fs 1     {Spring force, L side, Axle 1 - klbs           }
71  R alph 1   {Slip angle, R side, Axle 1 - deg              }
```

```
72  R Fz 1      {Load, R side, Axle 1 - klbs                    }
73  R Fy 1      {Side force, R side, Axle 1 - klbs              }
74  R Mz 1      {Aligning moment, R side, Axle 1 - ft-lbs       }
75  R Fs 1      {Spring force, R side, Axle 1 - klbs            }
76  phi Ax 2    {Roll, Axle 2 - deg                             }
77  Z Axle 2    {Bounce, Axle 2 - ft                            }
 •
 •
 •
```

9.3 Channel Names from the Phase-4 Model

The following list indicates the spelling used for the short names, long names, and unit names for most of the channels. The channel numbers are for a specific vehicle configuration (this was a 13-axle triples combination) and will not be the same for other configurations. (The list was obtained with the ERDP plotting program. The short name is given first, followed by the long name and units in the curly brackets.)

Input

```
 1  Steer in    {Input Steer Angle - deg                        }
 2  Brake in    {Brake Treadle Pressure - psi                   }
```

Hitch Variables

```
 3  Art 1-2     {Articulation Angle, Hitch 1-2 - deg            }
 4  Art 2-3     {Articulation Angle, Hitch 2-3 - deg            }
 5  Art 3-4     {Articulation Angle, Hitch 3-4 - deg            }
 6  Art 4-5     {Articulation Angle, Hitch 4-5 - deg            }
 7  Art 5-6     {Articulation Angle, Hitch 5-6 - deg            }
```

Sprung Mass Variables

```
 8  X cg #1     {X Position, cg, Tractor - ft                   }
 9  Y cg #1     {Y Position, cg, Tractor - ft                   }
10  Z cg #1     {Z Position, cg, Tractor - ft                   }
11  Roll #1     {Roll Angle, Tractor - deg                      }
12  Yaw #1      {Yaw Angle, Tractor - deg                       }
13  Pitch #1    {Pitch Angle, Tractor - deg                     }
14  u cg #1     {X Velocity, cg, Tractor - ft/sec               }
15  v cg #1     {Y Velocity, cg, Tractor - ft/sec               }
16  w cg #1     {Z Velocity, cg, Tractor - ft/sec               }
17  p of #1     {Roll Rate, Tractor - deg/sec                   }
18  r of #1     {Yaw Rate, Tractor - deg/sec                    }
19  q of #1     {Pitch Rate, Tractor - deg/sec                  }
```

```
20  Ax cg #1   {Long. Accel., Tractor - g's                }
21  Ay cg #1   {Lat. Accel., Tractor - g's                 }
22  Slip #1    {Slip Angle, Tractor - deg                  }
23  Rho cg#1   {Curvature, Tractor - 1/ft                  }
```

Axle Variables (single-digit axle numbers)

```
24  Z Axle 1   {Bounce, Axle 1 - ft                        }
25  Phi Ax 1   {Roll, Axle 1 - deg                         }
26  X Axle 1   {X Position, Axle 1 - ft                    }
27  Y Axle 1   {Y Position, Axle 1 - ft                    }
28  Xroll 1    {Auxiliary Roll Moment, Axle 1 - ft-lbs     }
```

Half-Axle Variables (single-digit axle numbers)

```
29  L Fz 1     {Load, L side, Axle 1 - klbs                }
30  L Fx 1     {Brake Force, L side, Axle 1 - klbs         }
31  L Fy 1     {Side Force, L side, Axle 1 - klbs          }
32  L Ux 1     {Long. Adhesion Ut., L , Ax 1 - --          }
33  L Uy 1     {Lat. Adhesion Ut., L side, Ax 1 - --       }
34  L Alph 1   {Slip Angle, L side, Axle 1 - deg           }
35  L Mz 1     {Aligning Moment, L side, Axle 1 - ft-lbs   }
36  L B/P 1    {Brake Pressure, L side, Axle 1 - psi       }
37  L B/T 1    {Brake Torque, L side, Axle 1 - ft-lbs      }
38  L Sx 1     {Longitudinal Slip, L side, Ax 1 - --       }
39  L W 1      {Spin Velocity, L side, Axle 1 - rad/sec    }
40  L Wdot 1   {Spin Acceleration, L side, Ax 1 - rad/s**2 }
41  L del 1    {Spring Deflection, L side, Ax 1 - in       }
42  L Fs 1     {Spring Force, L side, Axle 1 - klbs        }
43  L Str 1    {Steer Angle, L side, Axle 1 - deg          }
44  R Fz 1     {Load, R side, Axle 1 - klbs                }
45  R Fx 1     {Brake Force, R side, Axle 1 - klbs         }
46  R Fy 1     {Side Force, R side, Axle 1 - klbs          }
47  R Ux 1     {Long. Adhesion Ut., R , Ax 1 - --          }
48  R Uy 1     {Lat. Adhesion Ut., R side, Ax 1 - --       }
49  R Alph 1   {Slip Angle, R side, Axle 1 - deg           }
50  R Mz 1     {Aligning Moment, R side, Axle 1 - ft-lbs   }
51  R B/P 1    {Brake Pressure, R side, Axle 1 - psi       }
52  R B/T 1    {Brake Torque, R side, Axle 1 - ft-lbs      }
53  R Sx 1     {Longitudinal Slip, R side, Ax 1 - --       }
54  R W 1      {Spin Velocity, R side, Axle 1 - rad/sec    }
55  R Wdot 1   {Spin Acceleration, R side, Ax 1 - rad/s**2 }
56  R del 1    {Spring Deflection, R side, Ax 1 - in       }
57  R Fs 1     {Spring Force, R side, Axle 1 - klbs        }
58  R Str 1    {Steer Angle, R side, Axle 1 - deg          }
```

```
 59  Z Axle 2   {Bounce, Axle 2 - ft                          }
  •
  •
  •
369  Slip #6    {Slip Angle, 3d Semi-trailer - deg           }
370  Rho cg#6   {Curvature, 3d Semi-trailer - 1/ft           }
```

Axle Variables (double-digit axle numbers)

```
371  Z Axle10   {Bounce, Axle 10 - ft                        }
372  Phi Ax10   {Roll, Axle 10 - deg                         }
373  X Axle10   {X Position, Axle 10 - ft                    }
374  Y Axle10   {Y Position, Axle 10 - ft                    }
375  Xroll 10   {Auxiliary Roll Moment, Axle 10 - ft-lbs     }
```

Half-Axle Variables (double-digit axle numbers)

```
376  L Fz 10    {Load, L side, Axle 10 - klbs                }
377  L Fx 10    {Brake Force, L side, Axle 10 - klbs         }
378  L Fy 10    {Side Force, L side, Axle 10 - klbs          }
379  L Ux 10    {Long. Adhesion Ut., L , Ax10 - --           }
380  L Uy 10    {Lat. Adhesion Ut., L side, Ax 10 - --       }
381  L Alph10   {Slip Angle, L side, Axle 10 - deg           }
382  L Mz 10    {Aligning Moment, L side, Axle 10 - ft-lbs   }
383  L B/P 10   {Brake Pressure, L side, Axle 10 - psi       }
384  L B/T 10   {Brake Torque, L side, Axle 10 - ft-lbs      }
385  L Sx 10    {Longitudinal Slip, L side, Ax 10 - --       }
386  L W 10     {Spin Velocity, L side, Axle 10 - rad/sec    }
387  L Wdot10   {Spin Acceleration, L side, Ax 10 - rad/s**2}
388  L del 10   {Spring Deflection, L side, Ax 10 - in       }
389  L Fs 10    {Spring Force, L side, Axle 10 - klbs        }
390  R Fz 10    {Load, R side, Axle 10 - klbs                }
391  R Fx 10    {Brake Force, R side, Axle 10 - klbs         }
392  R Fy 10    {Side Force, R side, Axle 10 - klbs          }
393  R Ux 10    {Long. Adhesion Ut., R , Ax10 - --           }
394  R Uy 10    {Lat. Adhesion Ut., R side, Ax 10 - --       }
395  R Alph10   {Slip Angle, R side, Axle 10 - deg           }
396  R Mz 10    {Aligning Moment, R side, Axle 10 - ft-lbs   }
397  R B/P 10   {Brake Pressure, R side, Axle 10 - psi       }
398  R B/T 10   {Brake Torque, R side, Axle 10 - ft-lbs      }
399  R Sx 10    {Longitudinal Slip, R side, Ax 10 - --       }
400  R W 10     {Spin Velocity, R side, Axle 10 - rad/sec    }
401  R Wdot10   {Spin Acceleration, R side, Ax 10 - rad/s**2}
402  R del 10   {Spring Deflection, R side, Ax 10 - in       }
```

```
403   R Fs 10    {Spring Force, R side, Axle 10 - klbs        }

404   Z Axle11   {Bounce, Axle 11 - ft                        }
405   Phi Ax11   {Roll, Axle 11 - deg                         }
  •
  •
  •
501   R del 13   {Spring Deflection, R side, Ax 13 - in       }
502   R Fs 13    {Spring Force, R side, Axle 13 - klbs        }
```

APPENDIX: VERSION 1.00 OF THE ERD FILE

In May 1987, the ERD file layout was simplified and renamed Version 2.00. New files begin with the statement ERDFILEV2.00, whereas older Version 1.00 files begin with the statement ERDFILEV1.00. The ERD file toolbox on MTS (see Section 4) will read either type of file, but will only write Version 2.00 files. Any software that makes use of the toolbox will handle either version. All of the programs described in this manual will handle either type automatically. Unless specified otherwise, any ERD files created are of the Version 2.00 type.

The Version 1.00 layout differs from Version 2.00 only slightly. Version 1.00 files require eight mandatory lines in the header, rather than the three mandatory lines of Version 2.00 files. None of these lines use keywords. Table 17 summarizes the required lines in the header of a Version 1.00 ERD file. The first line identifies the file, the second line contains a title, and the third line is similar to the second line of a Version 2.00 file. The next five lines contain data associated with keywords GAIN, OFFSET, SHORTNAM, LONGNAME, and UNITSNAM. Additional lines follow that are identical to the optional lines in a Version 2.00 file. Version 1.00 files do not use an END statement to indicate the end of the header. Instead, the size of the header is specified in line 3 with the NXLINE parameter.

Prior to the Version 2.00 file, the "LINE 2" keyword used by the toolbox to access the parameters NCHAN, NSAMP, etc., did not exist. Instead, a keyword "LINE 3" was used. The "LINE 3" keyword is still supported by the toolbox. It is the same as "LINE 2" for indices of 1 and 2, to obtain NCHAN and NSAMP. An index of 3, needed to obtain NXLINE is meaningless. Indices of 4 through 8 correspond to indices of 2 through 7 with the "LINE 2" keyword. For example,

```
PINT (1, 5, 'LINE 3', NBYTES)
```

has the same effect as

```
PINT (1, 4, 'LINE 2', NBYTES).
```

Table 17. Summary of Records in a Version 1.00 ERD File Header.

Line No.	*Description*
1	*ERDFILEV1.00* — identifies file as having ERD format, version 1.00.
2	**Title** (80 characters, left-justified and padded with blanks)
3	NCHAN, NSAMP, NXLINE, NRECS, NBYTES, KEYNUM, STEP, KEYOPT — use commas to separate numbers NCHAN [integer] = Number of data channels NSAMP [integer] = Number of samples for each channel. The total number of sampled values in the data portion of the file is NCHAN × NSAMP. (If unknown, use –1.) NXLINE [integer] = Number of additional lines after line #8. NRECS [integer] = Number of records of data. (record ≈ line) Ignored for text data (KEYNUM = 5.) (If unknown, use –1.) NBYTES [integer] *Binary data:* Number of bytes/record. Should be chosen such that each record begins with channel 1: that is, NBYTES = K × NCHAN × B, where K is an integer and B is the number of bytes/number (B=2 for integer, B=4 for floating-point). *text data:* Number of samples/record. Thus each record contains NBYTES × NCHAN numbers. KEYNUM[integer] = Indicates how the data are stored. 0 = 2-byte integer (binary), 1 = 4-byte floating point (binary), 5 = Formatted floating-point (text) The format must be specified using the FORMAT keyword. STEP [real] = sample interval (e.g., time step) KEYOPT [integer] = number used in some data processing applications.
4	**Gains** (scale factors) for each channel, separated by commas [real]. Use 1.0 (for each channel) if data values are already scaled correctly.
5	**Offsets** for each channel, separated by commas [real]. Use 0.0 (for each channel) if data values are already scaled correctly.
6	**Short names** for all channels. [8 characters/name, left-justified]
7	**Long names** for all channels. [32 characters/name, left-justified]
8	**Unit names** for all channels. [8 characters/name, left-justified]
(9 to 8+ NXLINE	**Optional records**. Each record begins with an 8-character keyword, followed by information that can be used by application programs that recognize the keywords. The number of optional records here must be specified in line 3 as NXLINE.
9+NXLINE	**First** data record.
8+NXLINE +NRECS	**Last** data record.