[illegible]
74162

UMTRI-86-52

An Enhanced Simulation Capability for Studying the Braking, Steering, and Ride of Commercial Vehicles.

MVMA Project #6163

Demonstration/Seminar Materials
October, 1986

Michael Sayers
Yoram Guy
Charles MacAdam
Patricia Dill
Paul Fancher

UMTRI The University of Michigan
Transportation Research Institute

Technical Report Documentation Page

1. Report No. UMTRI-86-52	2. Government Accession No.	3. Recipient's Catalog No.
4. Title and Subtitle AN ENHANCED SIMULATION CAPABILITY FOR STUDYING THE BRAKING, STEERING, AND RIDE OF COMMERCIAL VEHICLES		5. Report Date October 1986
		6. Performing Organization Code
7. Author(s) M. Sayers, Y. Guy, C. MacAdam, P. Dill, P. Fancher		8. Performing Organization Report No. UMTRI-86-52
9. Performing Organization Name and Address The University of Michigan Transportation Research Institute 2901 Baxter Road Ann Arbor, MI 48109		10. Work Unit No. (TRAIS)
		11. Contract or Grant No. MVMA Proj. #6163
12. Sponsoring Agency Name and Address Motor Vehicle Manufacturers Association 300 New Center Building Detroit, MI 48202		13. Type of Report and Period Covered Final 7/85 - 6/86
		14. Sponsoring Agency Code
15. Supplementary Notes		

16. Abstract

This report is a discussion of recent changes and additions made to vehicle simulation programs to enhance their performance. It includes a collection of three independent sections written to document (1) the implementation of a preprocessor for the following UMTRI simulation programs: Phase 4, Yaw/Roll, Static Roll, Simplified Braking, and Linear Yaw, (2) the definition of output files and the development of a plotting capability for the outputs of simulations, and (3) the selection of formats for subroutines that may be used to include adaptive features in the simulation.

17. Key Words ERD files, preprocessor, system vehicle component files, Phase 4, adaptive simulation controls	18. Distribution Statement UNLIMITED

19. Security Classif. (of this report) NONE	20. Security Classif. (of this page) NONE	21. No. of Pages 128	22. Price

PREFACE

This document was distributed to the members of the Technical Advisory Panel on Braking and Handling of the Motor Truck Division of the Motor Vehicle Manufacturers Association (MVMA) at a Demonstration and Seminar held on June 25, 1986. Per agreement with the Panel, this material constitutes the final reporting effort on MVMA Project Number 6163 entitled "An Enhanced Simulation Capability for Studying the Braking, Steering, and Ride of Commercial Vehicles."

The main body of this document consists of three independent sections. The following persons from the Engineering Research Division of The University of Michigan Transportation Research Institute took primary responsibility for the work as indicated below:

Yoram Guy and Patricia Dill - Handling of Input Data for Computer Simulations
Michael Sayers - Filing and Presenting Results from Simulation Runs
Charles MacAdam - Interfacing Adaptive Control Features

The above persons were the authors of three independent sections documenting their work.

TABLE OF CONTENTS

INTRODUCTORY REMARKS

During this project, the following activities have been completed: (a) the implementation of a preprocessor for the following UMTRI simulation programs: Phase 4, Yaw/Roll, Static Roll, Simplified Braking, and Linear Yaw, (b) the definition of output files and the development of a plotting capability for the outputs of simulations, and (c) the selection of formats for subroutines that may be used to include adaptive features in the simulation. The overall project scope is illustrated in Figure 1. It displays how the interaction of the various segments mentioned above creates an enhanced simulation capability for the studying of braking, steering, and ride of commercial vehicles.

THE PREPROCESSOR SYSTEM

The preprocessor consists of a structured set of files and a system for translating the contents of these files into data sets and execution-control commands. These data sets and execution commands are used by the programs, specified by the users, in simulating the steering and braking performance of vehicles. The system for operating the simulations contains an INTERFACE PROGRAM (Figure 2) which uses the information contained in a SIMULATION FILE to construct an INPUT FILE (data set) that will be used to make a simulation run. The dashed lines in Figure 2 indicate the "data flow" involved in making a simulation run and obtaining OUTPUT FILES.

Given a system of this type, the process of making a simulation run becomes user friendly. A computerized system assembles an input file for the desired simulation program, rather than requiring the user to become proficient in assembling input files in program-specific format.

The power of the preprocessor system comes from having sets of files that describe vehicle components and maneuvers that are of interest. The SIMULATION FILE contains directions as to which components and maneuvers are to be used in a simulation run. The user of the system builds a SIMULATION FILE by figuratively "pointing to" the files that describe the vehicle and maneuver that is to be simulated. The "points to" property is illustrated by the solid interconnecting lines shown in Figure 2. Since the description of the vehicle involves pointing to several components and vehicle properties, a VEHICLE FILE–INDEX is built to describe pertinent vehicles, thereby allowing the SIMULATION FILE to only have to point to the desired VEHICLE FILE-INDEX.

The SIMULATION FILE also points to options that control the type of computer model to be used and the output format for that model (Figure 2). The INTERFACE PROGRAM has been arranged to communicate with several of our models in order to consolidate the models and improve the compatibility between models.

The preprocessor system is operational and has been tested. Sizeable sets of suspension, spring, and tire files (corresponding to components that we have measured previously) have been assembled. Examples of the other types of files employed in describing vehicles have been created and VEHICLE FILE-INDEXs for common types of vehicles have been built.

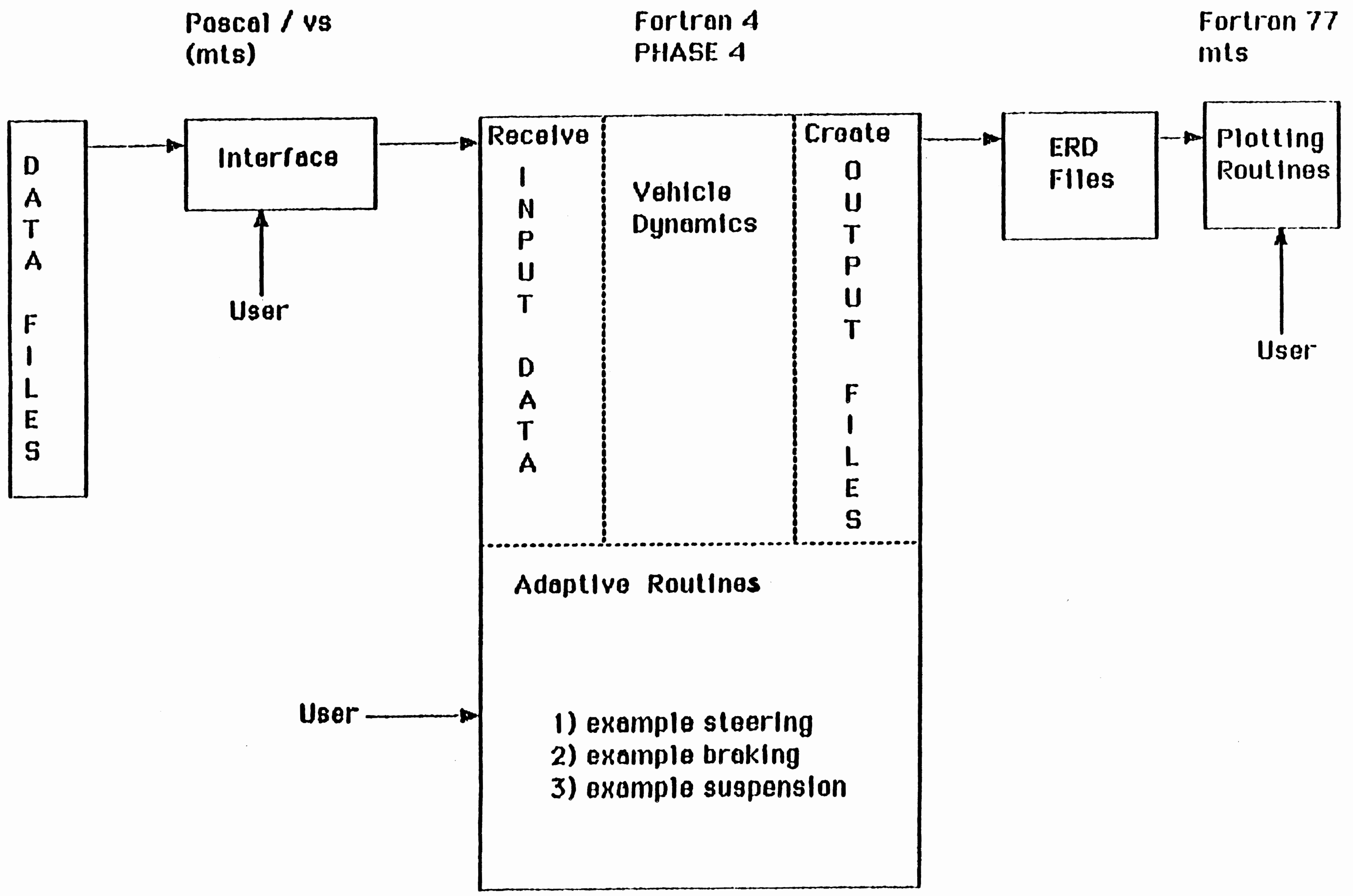


Figure 1. Current Overview

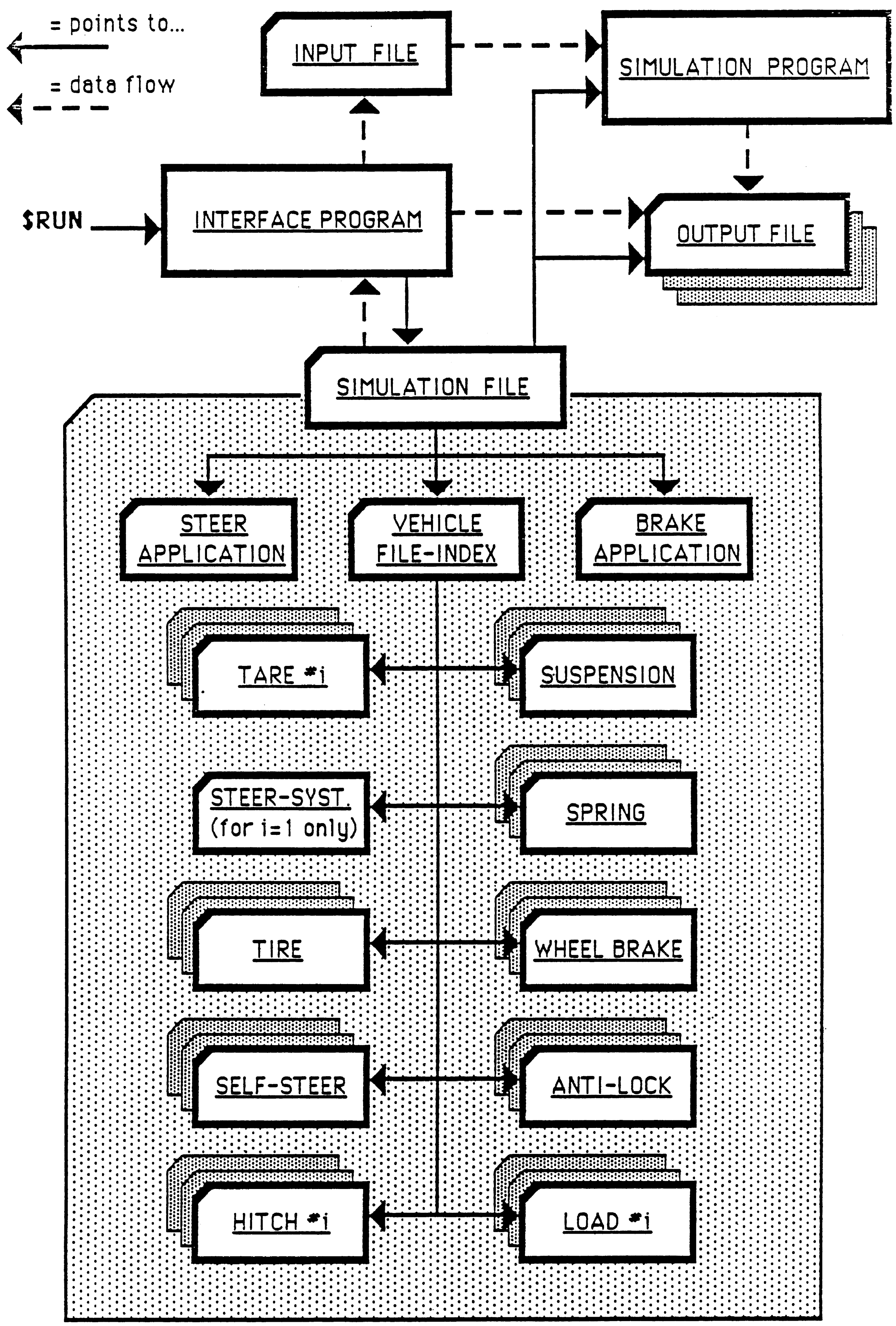


Figure 2. Preprocessor System

THE SYSTEM FOR HANDLING THE OUTPUT

The work on improving the handling of the output data from the Phase 4 and Yaw/Roll simulations has resulted in the development of special output files and a means for graphically displaying the contents of these files.

The Phase 4 and Yaw/Roll models have recently been extended to produce files containing time histories of simulation variables. After a simulation run, these files, referred to as "ERD files," contain numerical data in the same form as a multi-channel tape recorder, and an extensive amount of labeling information.

A program called "ERDP" is now available to plot results from any ERD-type output file. Given ERD files of simulation time histories, the ERDP program can display the results in a variety of formats.

PROVISIONS FOR ADAPTIVE CONTROLS

Instead of developing one subroutine for all types of controls, it was found to be more convenient and reasonable to develop three subroutines—one that controls the steer angles of any of the wheels, a second that applies "vertical" forces at any suspension reference point on either side of the vehicle, and a third that modulates brake chamber pressures. Simple control algorithms have been chosen to demonstrate how these features might be used.

NATURE OF THE CONTENTS

This report is a collection of three independent sections written to document various components of the MVMA project to develop an enhanced simulation capability for studying the braking, steering, and ride of commercial vehicles. Since each section addresses a different topic within the project, the individual sections are not related to each other directly.

Preprocessor System

1.0 INTRODUCTION AND GENERAL INFORMATION

1.1 INTRODUCTION

The UMTRI Simulation Interface is a system that allows different computer simulations to be run on the same vehicle. Each simulation requires an input file that describes the vehicle being simulated and data about the desired path of the vehicle. This information is required in different forms by each of the simulations. The interface program accepts input which designates the components of the vehicle to be simulated. Regardless of the simulation model to be used, the interface extracts data from the same vehicle component files and uses the same type of calculations to construct each simulation input file. Thus, each simulation is essentially acting upon the same vehicle, producing results that can be compared directly. The simulations that are currently available on the UMTRI Simulation Interface are the Phase 4 Model, Yaw Roll Model, Simplified Braking Model, Static Roll Model, and the Linear Yaw Plane models.

1.2 ENGINEERING UNITS AND COMPUTER REQUIREMENTS

Throughout the Simulation Interface, the English system of units is used. With the exceptions listed in the vehicle component files, all input data are given in units of pounds, inches, degrees, and seconds. Masses and weights are in units of pounds with a gravitational constant of 386 in/sec/sec assumed. (For specific information see the general notes about vehicle component files in Section 3.3.)

The simulations accessed by the interface can be used on any large scale computer system. (Please see the appropriate simulation's user's manual for specifics on computer requirements.) The Simulation Interface as a whole was designed to take advantage of certain system specific properties of the Michigan Terminal System. It has not yet been tested on any other type of computer system.

This system was designed to facilitate the study of the influence of weights and dimensions on vehicle stability and control in the RTAC Canadian Truck study. Information about this system may be obtained by contacting the Engineering Research Division at the Transportation Research Institute, University of Michigan, 2901 Baxter Road, Ann Arbor, Michigan, 48109.

2.0 USE OF THE SIMULATION INTERFACE

2.1 GETTING STARTED - RUNSIM

All of the files and programs needed to run the simulation interface are stored on the computer account ST6T on the MTS computer system. In order to run this system, the user must create a SIMULATION file as specified in section 3.3, which indicates to the system which vehicle is to be simulated, and which simulation is to be performed.

If the simulation interface is to be run from a computer account other than ST6T, the following commands must be included in that account's sigfile:

```
$set macros=on
$>set var maclib(1)="st6t:runsim.mac"
```

in that order. Also, that account must have read access for all of the files in the account ST6T.

2.1.1 Input

The entire simulation interface can be run by using the macro RUNSIM. This macro allows all the programs and file operations to be accomplished with the user issuing only one command.

```
$RUNSIM Simulation_file
```

2.1.2 Output

The main output from the simulation interface is stored in two files, a print file and an ERD file. These files are named according to the naming conventions discussed in Section 3.3. The print file contains output from the simulation itself, plus a page of output from the Post Processing program, which makes a short analysis of the simulation output (as discussed in section 4.4). This print file is automatically sent to the printer and delivered to UMTRI. The ERD file contains a binary representation of the simulation run in a specialized format that is discussed in Mike Sayers' memos "ERD Files : Format and Layout" and "ERD File Outputs from the Vehicle Simulations."

2.1.3 Possible Problems and Solutions

- Program not found or access not allowed - When this type of error is encountered, you must immediately abort the macro by hitting the control and the E keys at the same time, a couple of times, until the # sign appears. If access was not allowed, then sign on to the account where the file is stored and permit it to your account. If the program was not found, then sign on to the account ST6T and *RESTORE that file. (See the MTS manual on Public Files for information about *RESTORE.)

- Component file not found - When this type of error occurs you have two choices. Either enter the correct file name at the ? prompt after the error, or Control E out of the macro. Once out of the macro you can check your simulation file for file name misspellings, or if the file has been destroyed, you can *RESTORE it.

- Access not allowed to component file - To correct this problem, control E out of the macro, sign on to the account where that file is stored, and permit it to your account.

- Program fails - When a program within the macro fails, you must control E out of the macro immediately, and check your component files for valid data. You may use the debug file from the interface program to check your data. (§3.2.7)

- Program exceeds local time limit - If there is a local time limit specified in the sigfile, you may want to increase it. If the time limit is sufficient for the run of the program, then you may be stuck in a loop in the simulation program. In that case it is best to look at the print file to see what has happened in the simulation. Note : If the time limit is exceeded, the print file will not be printed automatically; you must copy it to the printer yourself.

- Run out of money in your account - When this error occurs, simply type RETURN to continue when prompted, to finish the interface run. If you are on a master account DO NOT sign off until you put more money in the account using the ACC Manager.

2.2 USE OF EACH PROGRAM SEPARATELY

Within the Simulation Interface System, there are three types of programs: the Interface program, the Simulations, and the Post Processor. These programs can be run individually, without using the RUNSIM macro. The sections below describe how to run each of these programs, the input required, and the output files produced.

2.2.1 Interface Program

The Interface Program may be run by issuing the command:

```
$Run VS.OBJ SCARDS=simulation_file SPRINT=*dummy* t=5
```

This command will run the interface program with simulation_file as its input. SPRINT=*dummy* prevents the numerous program messages from being printed on the screen, and t=5 is the CPU time limit on the program.

<u>Input</u> : This program takes as input a simulation_file that is described in detail in section 3.3. It is submitted on the logical I/O unit SCARDS, and indicates which vehicle is going to be simulated, and which simulation input file to prepare.

<u>Output</u> : There are three major output files created by this program, destination, debug.out, disp.out. The destination file contains the simulation input file that the program creates. The debug.out file contains a listing of steps the program goes through, variable assignments, and component files read. The disp.out file is a list of read and calculated vehicle parameters, and a record of component file assignments.

<u>Problems and Solutions</u> : Most problems that can arise in this program have to do with the component files used. If a component file is not found by the program, or if access to that file is not allowed, then follow the steps outlined in Section 2.1.3. If the problem is a destination file that is not correct, then an inspection of the debug.out and the disp.out files is in order. The disp.out file is used as a debugging and engineering tool, and displays the static properties of the vehicle in both the empty and loaded conditions.

The debug.out file is a useful debugging device because it records the last steps taken by the program before aborting, if the program fails. This information can provide some clue as to where the program went wrong.

2.2.2 Simulation Programs

There are four computer simulations that are supported by the UMTRI Simulation Interface System:

Phase 4 Model
Yaw Roll Model
Simplified Braking Model
Static Roll Model
Linear Yaw Plane Models

Detailed information about each of these models may be obtained by referring to the appropriate user's manual. A general description of each model may be found in Section 3.4.

Input : Each of these models takes as input the output file from the interface program, destination. The correct simulation input file is created by indicating which simulation is to be performed in the simulation file. Each input file is created according to the format described in the appropriate user's manual.

Output : Each of these models creates as output two main files, the print file and the ERD file. The print file contains an echo of the output to the simulation, as well as a record of the simulation. This file is a text file and may be copied to the printer. The ERD file is a binary record of the simulation and cannot be viewed on the screen or copied ot the printer. It is used as input to the post processor program, and for plotting purposes. See Mike Sayers' memo "Plot : A Plotting Subroutine for Engineering Applications" for more information about plotting simulation data.

How to Run Each Simulation : Each of the simulations has one input file, destination, and at least two output files, printfile and ERD_file. Note: The ERD_file has not yet been implemented in the Static Roll Model. There are some output units that are not necessary for our purposes; these units are assigned to *dummy*. The scientific subroutines packages naas:eispack and naas:nal are used by three of the simulations and must be added to the run command. The run commands are listed below.

> Phase 4 Model

```
$Run ST6T:Ph4.RTAC.O+naas:eispack+naas:nal 2=ERD_file 5=destination
  6=printfile t=60
```

>Yaw Roll Model

```
$Run ST6T:Yaw.RTAC.O+naas:eispack+naas:nal 2=ERD_file 5=destination
  6=printfile t=60
```

>Simplified Braking Model

```
$Run ST6T:Brake.RTAC.O+naas:eispack+naas:nal 2=ERD_file 5=destination
  6=printfile t=5
```

>Static Roll Model

```
$Run ST6T:SRoll.RTAC.O 5=destination 6=printfile 7=*dummy* t=20
```

>Linear Yaw Plane Models (X represents the number of units. Tractor and trailer equals 2)

```
$Run SVDA:Rao.EigenXo+naas:eispack+naas:ssp 5=destination 6=printfile t=10

$Run SVDA:Rao.FreqXo+naas:eispack+naas:ssp 5=destination 6=printfile t=10

$Run SVDA:Rao.TimeXo+naas:eispack+naas:ssp 5=destination 6=printfile t=10

$Run SVDA:Rao.PzeroXo+naas:eispack+naas:ssp 5=destination 6=printfile t=10
```

2.2.3 Post Processor Program

The Post Processor program analyzes the ERD_file created by a simulation program. This analysis may consist of a combination of up to eight different vehicle performance measures. These measures include: A1-static rollover threshold, A2-yaw stability, A3-high-speed offtracking, B-response to rapid steering reversals, C1-low-speed offtracking, C2-tight turn jackknife conditions, D-braking in a turn, E-braking efficiency. For a more detailed description of these measures, refer to Section 5.4.

Input : The Post Processor takes as input two files, the ERD_file and the file RUNFILE. The ERD_file is the output from a computer simulation run attached to the logical I/O unit 2. The RUNFILE is a file created by the user to indicate to the Post Processor program which performance measures are to be calculated. This file must be named RUNFILE, and appear in the account from which the program is being run. It is a text file that contains the letter codes of each measure desired, separated by commas. For example, the line A1,A2 would instruct the program to calculate the static rollover threshold and the yaw stability of the vehicle. Note: Entering the letter A alone, specifies that measures A1, A2, and A3 are to be calculated, and similarly, C=C1+C2. This program can be executed by issuing the command :

```
$Run ST6T:Post.obj scards=ERD_file t=30
```

Output : The output from this program appears in the file measures.out. This is a text file that contains the results of the performance measure analysis that was calculated. When the Simulation Interface System is run using the macro RUNSIM, this file is attached to the print file and copied to the printer. When the programs are run separately, this file, measures.out, must be copied to the printer manually. This is accomplished by issuing the command : $copy measures.out *print*.

2.2.4 RTAC Database

The RTAC database is a file that stores the results of all the vehicle performance measures calculated by the Post Processor program. The database is arranged according to the vehicle configurations and variations outlined in the simulation matrix. Each vehicle record contains information that describes the vehicle, and the results of all the vehicle performance measures calculated for that vehicle. The information is stored in binary records. When a new performance measure is calculated by the Post Processor program, the Post Processor creates a file named measure.temp with this new information in it. This information is incorporated into the RTAC database by running the Database Updating program. This program can be run by issuing the command:

```
$Run meas.updat.o t=10
```

The Database Updating program adds the new performance measures to the appropriate vehicle record in the database file, RTAC.Data.

Examination of the RTAC Database is accomplished using the Microsoft Excel program on an Apple Macintosh computer. Since the database file is in the form of binary records, it needs to be translated to a text format before it can be examined. To do this, run the program:

```
$run Excel.obj t=10
```

Once the database is converted to text, it has to be converted to "Excel" format, in order to allow it to be read by the program Excel. To do this, edit the file RTAC.Data, and replace every occurrence of @| with a tab. The Database is now ready to be downloaded to an Apple Macintosh and examined using Excel.

3.0 COMPONENTS OF SIMULATION INTERFACE

In this section, each part of the simulation interface system will be briefly described . In particular, how each segment works, how each is set up, and what each segment's purpose is in the system will be discussed.

3.1 RUNSIM MACRO

The macro Runsim can be used to run the entire Simulation Interface System. A macro is a special set of MTS system programs that act as an interface between the user and MTS. It is essentially a file of MTS commands and command extensions that can be run. A macro is quite similar to a source file, but is much more powerful due to the command extensions allowed. More information on the use and scructure of macros is available in the Computing Center's manual on "Command Extensions and Macros."

In Section 2.2, the use of each component in the Simulation Interface was discussed. The purpose of the Runsim macro is to link these programs together by allowing communication among the files through text files, and it also automatically processes the output files as specified by the user in the simulation file. Thus, the sequence of steps necessary to generate simulation output are automated, and reduced to one command.

3.2 INTERFACE PROGRAM

The Interface program was designed to create input data files for the vehicle simulation programs included in the system (Phase 4, Yaw Roll, Static Roll, Simplified Braking, and the Linear Yaw Plane models). It creates the input file according to the type of simulation indicated and the vehicle selected in the Simulation file, which is the input file to the Interface program. The program extracts data about the vehicle from the vehicle component files specified in the description of the vehicle in the Index file. All of the component files are stored in the Vehicle Component Library on the MTS computer account ST6T. These files are available to any account that is authorized to use the UMTRI Simulation Interface System.

3.3 VEHICLE COMPONENT FILES

GENERAL NOTES ON PARAMETER-FILE SYNTAX, STRUCTURE, AND CONVENTIONS :

1. KEYWORDS (syntax literals) are shown and must be input in UPPER CASE.

2. The following characters [ ] { } / / ( ) are used below merely for syntax specification, and are not part of the actual parameter-file syntax :
   a. [Brackets] enclose optional input (not mandatory to enter).
   b. {Braces} enclose notes or comments, which are not part of file structure.
   c. /Slashes/ are used to indicate mutually exclusive input alternatives:
      (i) Between / KEYWORDS - to separate between alternate literals of one mandatory input keyword on the line.
      (ii) / Leading & trailing / on line(s) - to delimit alternate line formats.

3. Actual order of lines is insignificant, except for the following:
   a. First 4 lines (a title line and 3 comment lines) are merely echoed, and otherwise ignored (except for b.).
   b. The title line of the Vehicle-Index file is stored as the Run-Title.
   c. A few specific line-order restrictions are indicated where applicable.

4. Parsed column range is 1 - 80, and file lines may be indented as desired.

5. All entries of each line are input sequentially in free format, adjacent entries being separated by any number and sequence of blanks and/or commas (" 2, ,1 " will cause the second entry to be read as 1 !).

6. Internal notes or comments in actual parameter file (in addition to 3.a.):
   a. To be echoed: May appear on any line in the file, to the right of the rightmost entry on the line.
   b. Not echoed : Any additional lines below the specified last input line.

7. A full trailer is considered as two units (a dolly and a semitrailer).

8. All x (length) dimensions are with reference to each unit's front articulation point (front axle CL for unit #1), positive rearwards.

9. Hitch-File defines 5th-wheel/turntable/pintle-hook/s between current Unit# and Unit#+1 (HITCH line for last unit is ignored, if entered).

10. Specific issues for the Vehicle-Index file :
    a. VEHICLE line must be fifth, and TRKTR sixth. A single STRSTM line and a single FSUSP line must precede the first RSUSP line in the file.
    b. Any RSUSP, HITCH & LOAD lines refer to the last preceding "tare" line (TRKTR/SEMI/DOLLY).
    c. Any SLFSTR, SPRING, TIRE, WHLBRK & ALOCK lines refer to the last preceding FSUSP or RSUSP line.
    d. Any RHS line modifies the indicated parameter (on the vehicle right handside only) for the last preceding FSUSP or RSUSP.
    e. Any RHS and/or DEFAULT keywords must precede the component keyword (SPRING, TIRE, etc.) of the line.
    f. A DEFAULT directive will cause the assignment of the data from that line also to all other occurrences of the same type (RSUSP, SPRING, etc.), which were not explicitly specified by separate lines.
    g. Any RHS line with a DEFAULT directive will work as described above, but on the vehicle right-handside only (RHS & DEFAULT positions on the line may be swapped, provided that rule d. is observed).
    h. All MTS file-names specified in the Vehicle-Index-File must feature their CCID: prefix, regardless of the CCID used for the simulation.

11. Physical Units : Unless otherwise specified in the prototype component files, English Engineering Units are assumed, as below:

Data Type	Unit
Vehicle and component dimensions	inches
Component translational deflection	inches
Yaw, Roll, Angular deflection	degrees
Trajectory path (X-Y)	feet
Velocity	ft/sec
Time	seconds
Weight, Load, Force	lbs.
Torque, Moment	in_lbs.
Moment of Inertia	in_lbs_sec^2
Translational stiffness	lbs./in
Angular stiffness	(in_lbs.)/deg
Pressure	psi

{PROTOTYPE SIMULATION FILE}

```
    Title
    {3 comment lines (text or blank)}
    VEHICLE, Vehicle-Index-File
    PROGRAM, NS / LY / SR / YR / P4 / SB, = / - / Program-Input-File
    [MEASURE, [[A] / [[A1,] [A2,] [A3,]]] [B,] [[C] / [[C1,] [C2,]]] [D,] [E]]      {Note
    1.}
    [STRAPL, PATH / ANGLE / LRANGLE, Steer-Applic-File]  {this +/or next line}
    [BRKAPL, [BHYST,] [BPROP,] [ALOCK,] Brake-Application-File]
    VELTIM, Velocity, Simulation-Time
    [/ ROAD, PLANAR, Long-Slope, Side-Slope /
    / ROAD, USER, Road-File /]                    {either ROAD line - for P4 only}
    [CPUTIME, CPU-Time-Limit]                     {default is set to 200 sec}
    [DROLL, Roll-Increment]                       {for programs SR, YR only}
    [PRINT, [DISP,] [VELO,] [ACCE,] [TIREX,] [BRAKES,] [TIREY,]...
      { on one line }   ...[UNSPRGM,] [BTEMP,] = / - / Print-File, Time-incr]
    [ERDFILE, [DISP,] [VELO,] [ACCE,] [TIREX,] [BRAKES,] [TIREY,]...
      { on one line }  ...[UNSPRGM,] [BTEMP,] = / - / ERD-Plot-File, Time-incr]
END
```

{ 1. See RTAC Simulation Plan, section 2.0, for explanation of performance measures. "A" stands for "A1"+"A2"+"A3", and similarly, "C"="C1"+"C2".

2. All MTS file-names specified in the Simulation-File <u>must</u> feature their CCID: prefix, regardless of the CCID used for the simulation.

3. An equal sign (=) entered instead of a full MTS file-name on either a PRINT, or ERDFILE line causes the relevant output to be written to an automatically-created permanent file, whose name is formed by the respective prefix 'PR.', or 'ER.', followed by characters # 4 thru 12 of the simulation-file name. An equal sign on a PROGRAM line will cause the assignment of the specified program code ('YR',P4', etc.) as the prefix in the name of the Program-Input-File, followed by characters #4 through 12 of the simulation-file name. For example, a simulation file ST6T:SI.TRSEMI.WT with the lines

```
        PROGRAM,YR,=
         ERDFILE, =
```

will create a permanent Yaw/Roll input-file YR.TRSEMI.WT, and later will direct the ERD-format simulation output data to a permanent file ER.TRSEMI.WT of the active CCID, from which the run was invoked.

4. A minus sign (-) entered instead of a full MTS file-name on either a PROGRAM, PRINT, or ERDFILE line causes the relevant output to be written to a temporary file (a leading -), whose name is formed by the same prefix as defined in 3, followed by characters # 4 thru 8 of the simulation-file name. For example, a simulation file ST6T:SI.TRSEMI.P4 with a line PROGRAM, P4, - will direct the Phase 4 input data to a temporary file -P4.TRSEM .

5. On PRINT line, optional output selection directives are effective only when running Phase 4 (ignored by other programs), and the absence of all optional output directives will invoke full output for all eight variable groups. }

{PROTOTYPE VEHICLE-INDEX FILE}

```
Title
{3 comment lines (text or blank)}
VEHICLE, Num-Of-Units                        { < 8 }
TRKTR, Tare-File                             {predefined as Unit#1}
    STRSTM, Steering-System-File
    FSUSP, Front-Suspension-File
        [SLFSTR, Selfsteer-File]
        [RHS,] [DEFAULT,] SPRING, LINEAR / TABLE / ENVLP, Spring-File]
        [[RHS,] [DEFAULT,] TIRE, LINEAR / TABLE / MODEL, Tire-File]
        [[RHS,] [DEFAULT,] WHLBRK, LINEAR / TABLE / MODEL, Whlbrake-
        File]
        [[RHS,] [DEFAULT,] ALOCK, Antilock-File]
    [DEFAULT,] RSUSP, Rear-Susp #, Suspension-File
        [SLFSTR, Selfsteer-File]      {identical properties assigned on tandems}
        [[REAR,] [RHS,] [DEFAULT,] SPRING, LINEAR / TABLE / ENVLP,
        Spring-File]
        [[REAR,] [RHS,] [DEFAULT,] TIRE, LINEAR / TABLE / MODEL, Tire-
        File]
        [[REAR,] [RHS,] [DEFAULT,] WHLBRK, LINEAR / TABLE / MODEL,
        Whlbrake-File]
        [[REAR,] [RHS,] [DEFAULT,] ALOCK, Antilock-File]
    [HITCH, Hitch-File]
    [LOAD, Load-File]
[SEMI, Unit#, Tare-File              {up to 3 SEMI lines allowed}
        {---> here need list only those suspension, spring, tire, brake and antilock files
        which are different than any indicated "DEFAULT"s.  See note}
        [SLFSTR, Selfsteer-File]      {RSUSP line must precede, if more than 1
        RSUSP}
    [HITCH, Hitch-File]
    [LOAD, Load-File]]
[DOLLY, Unit#, Tare-File             {up to 3 DOLLY lines allowed}
        {---> here need list only those suspension, spring, tire, brake and antilock files
        which are different than any indicated "DEFAULT"s.  See note}
        [SLFSTR, Selfsteer-File]      {RSUSP line must precede, if more than 1
        RSUSP}
    [HITCH, Hitch-File]
    [LOAD, Load-File]]
END
```

{Note : Must enter a null component file, when DEFAULT is in effect but a component has to be omitted (such as no brakes on an axle, while DEFAULT WHLBRK is specified elsewhere in the index file)}

{PROTOTYPE SPRING FILE}

```
Title
{3 comment lines (text or blank)}
SPRING
[LINEAR, Rate]
[FRICT, Coulomb-Friction]
[TABLE, Table-Lines
 Force, Deflection]                {Table-Lines}
[ENVLP, One-Way-Lines
 JOUNCE, Jounce-Beta
 Force, Deflection                 {One-Way-Lines}
 REBOUND, Rebound-Beta
 Force, Deflection]                {One-Way-Lines}
END
```

{ Note: For a walking-beam tandem-suspension spring, the force values are entered "per wheel", not "per side" ! ! }

{PROTOTYPE SUSPENSION FILE}

```
Title
{3 comment lines (text or blank)}
/ FSUSP /
/ RSUSP, Susp-Key /
INERT, Unsprung-Mass, Ixx                                { per axle ! }
[LONGL, Axle-Sep, Static-Load-Dist, Dynamic-Load-Trans] {for Susp-Key>0}
VERT, Axle-CG-Height, Roll-Center-Height
TRACK, Track, Duals-Sep, Spring-Spread
ROLL, Aux-Roll-Stiffness, Rollsteer-Coef
[VISC, Lhs-Damping-Coef, Rhs-Damping-Coef]
[REAR, INERT, Unsprung-Mass, Ixx]                        { per axle ! }
[REAR, VERT, Axle-CG-Height, Roll-Center-Height]
[REAR, TRACK, Track, Duals-Sep, Spring-Spread]
[REAR, ROLL, Aux-Roll-Stiffness, Rollsteer-Coef]
[REAR, VISC, Lhs-Damping-Coef, Rhs-Damping-Coef]
END
```

{ Note: Susp-Keys: 0 - Single ; 1 - Four Spring Tandem ; 2 - Walking Beam Tandem.
REAR directive valid only for Susp-Key > 0, and redundant for identical leading and trailing axle data. }

{PROTOTYPE HITCH FILE}

```
Title
{3 comment lines (text or blank)}
HITCH, Hitch-Type                    {1 - 5th Wheel, 2 - Inverted 5th wheel,
       3 - Compensating 5th wheel, 4 - Turntable, 5 - 'A' dolly, 6 - 'B' dolly}
[HROLL, Roll-Center-Height]          {above 5th wheel plane - for type 3 only}
[SEPAR, Sep-Moment, Sep-Angle]       {used by static roll model only}
[KINEM, Hitch-GLA, Hitch-KLA]        {used by CW yaw roll model, for type 6
only}
[LINEAR, Yaw-Stiffness, Roll-Stiffness] {only for special YR runs, or type 6}
[LASH, Yaw-Lash-Angle, Roll-Lash-Angle] {from here down - for type 6 only}
[VISC, Yaw-Damping, Roll-Damping]]
[TABLE                               { not yet implemented ! ! }
[YAW, Yaw-Stiff-Lines
Z-Moment, Z-Angle]                   {Yaw-Stiff-Lines}
[ROLL, Roll-Stiff-Lines
X-Moment, X-Angle]]                  {Roll-Stiff-Lines}
END
```

{PROTOTYPE TARE FILE}

```
Title
{3 comment lines (text or blank)}
/ DOLLY, Sprung-Mass /
/ TRKTR, Sprung-Mass
  TFRAME, Torsional-Stiffness, Torsional-Friction, Torsion-Axis-Height /
/ SEMI, Sprung-Mass
  KINGPIN, KP-Setting /
[BEDXYZ, Bed-Length, Bed-Width, Bed-Floor-Height] {for truck & semi only}
RSUSP, Num-Of-Rear-Susps
 Suspension-Num, Wheelbase-to              {Num-Of-Rear-Susps lines ;  Note 1}
CGXYZ, Sprung-CG-Dist, Sprung-CG-Offset, Sprung-CG-Height
INERT, Sprung-Ixx, Sprung-Iyy, Sprung-Izz
HITCH, X-Location, Y-Location, Z-Location  {Notes 1, 2}
END
```

{ Notes:

1. "Wheelbase-to", and HITCH "x-Location" are, respectively, the longitudinal distances of the suspension's C.L. and the rear hitch C.L.measured from front axle on tractors/trucks, from front articulation point on dollies and semitrailers.
2. HITCH locations refer to 5th-wheel/turntable/pintle-hook/s on given Unit# (HITCH line for last unit is redundant - ignored if entered).}

{PROTOTYPE WHEEL BRAKE FILE}

```
Title
{3 comment lines (text or blank)}
WHLBRK, Time-Lag, Rise-Time
[LINEAR, Torque-Coef]
[TABLE, Table1-Length
  Pressure, Torque]          {Table1-Length lines}
[MODEL
  Chamber-Area, Drum-Diamtr, Wedge-Angle/Arm-Length, Pushout-Pressure
  Bfo, Cv, Ct, Cf            {brake factor coefficients in eq. I.2.3, p.321, Ph IV
  manual}
  BTEMP, Table2-Length
  Init-Temp, Temp-Coef   {Table2-Length lines}
  Drum-Rub-Area, Drum-Thickness, Drum-Convect-Coef
  Drum-Temp, Ambient-Temp
  Lining-Area, Lining-Thickness, Lining-Convect-Coef, Lining-Temp]
{.................................................................................}
[BHYST, Hy                   {if Hy = 0. then next line redundant (ignored if entered)}
  [Hy2, Resbrk, Resid, Hyl]]
[BPROP, Ipro [,Spring0  {for Ipro = 2 only}]     {see pp.56, 351, Ph IV manual}
  TREADLE, Table3-Length
  Press-Treadle, Press-Out {Table3-Length lines}
  VALVE, Table4-Length
  V-Ipro, K]                 {Table4-Length lines  -  see p. 348, Ph IV manual}
END
```

{PROTOTYPE LOAD FILE}

```
Title
{3 comment lines (text or blank)}
/ PAYLOAD, Payload-Weight /
/ SSPLOAD, Num-Of-SuspLoads
  Susp#, Susp-Load /                      {Num-Of-SuspLoads lines}
[DENSITY, Freight-Density]                { in units of lb/ft3 - see note 1 }
[INERT, Ixx, Iyy, Izz]                    {notes 2, 3 }
[CGXYZ, CG-Dist, CG-Lateral-Offset, CG-Height] {notes 3 thru 6 }
END
```

{ Notes:

1. DENSITY may be used in order to have the Payload CG height calculated based on the Freight-Density, and the cargo bed floor area and height.
2. INERT line is optional - if absent, then the Payload moments of inertia will be calculated based on a rectangular box of uniform density, the given or calculated Payload mass, the given or calculated Payload CG height, and the cargo-bed dimensions.
3. CGXYZ & INERT are net payload parameters.
4. CG-Dist is measured along x-axis, positive aft of front articulation point (front axle for Unit #1, king pin for a semi, pintle hook for a dolly).
5. CG-Height is with reference to ground, and ignored if a DENSITY line is entered.
6. If SSPLOAD is specified, then CG-Dist is normally ignored, but some best-estimate value must always figure on the line (may be used by program, if suspension loads are insufficient to solve for payloads and hitchloads–such as in B-trains, for next-to-last semi's).
7. LOAD files listed in a given Vehicle-Index file must all be of the same type (either all PAYLOAD or all SSPLOAD).
8. When PAYLOADs are specified, LOAD files are not mandatory for units with zero payload, but when SSPLOADs are specified, LOAD files are mandatory for all units. }

{PROTOTYPE TIRE FILE}

```
Title
{3 comment lines (text or blank)}
TIRE, Radius, Iyy
STIFFYZ, Lateral-Stiffness, Vertical-Stiffness
CAMBER, Camber-Stiffness, Overturning-Stiffness
[ALIGN, Aligning-Torque-Stiffness]           {not required by Model}
[CLONGL, Longitudinal-Stiffness]             {not required by Model or Table}
[CALFA, Cornering-Stiffness]                 {not required by Model or Table}
[PEAKMU, Peak-Cornering-Friction-Coefficient] {required by YR only}
{                                                                         }
[TABLE                        { Sequence of lines within TABLE part is fixed ! ! }
  CALFA, Num-Vert-Loads, Num-Velocities
  Vert-Load1 [, Vert-Load2] [, Vert-Load3]
  Velocity1 [, Velocity2] [, Velocity3]
  {-----------------------------------------------------------------------------
    one [---] block for each Calfa load-velocity combination}
    Load#, Velocity#, Length1
    Alfa, Mu-Y                 {Length1 lines;  increasing, positive Alfa !
  -----------------------------------------------------------------------------}
    ROLLOFF, M-Slip-Points [, N-Alfa-Points
                               {M-Slip-Points = 0 indicates no roll-off}
    Long-Slip1 [, Long-Slip2]...[, Long-SlipM]
    Alfa1 [, Alfa2]...[, AlfaN]
    Rolly11 [, Rolly12]...[, Rolly1M]
    {  :        :        :        -   N x M matrix}
    RollyN1 [, RollyN2]...[, RollyNM]]
  CLONGL, Num-Vert-Loads, Num-Velocities
  Vert-Load1 [, Vert-Load2] [, Vert-Load3]
  Velocity1 [, Velocity2] [, Velocity3]
  {-----------------------------------------------------------------------------
    one [---] block for each Clong load-velocity combination}
    Load#, Velocity#, Length2
    Slip, Mu-X                 {Length2 lines;  increasing, positive Slip !
  -----------------------------------------------------------------------------}
    ROLLOFF, M-Slip-Points [, N-Alfa-Points
                               {M-Slip-Points = 0 indicates no roll-off}
    Long-Slip1 [, Long-Slip2]...[, Long-SlipM]
    Alfa1 [, Alfa2]...[, AlfaN]
    Rollx11 [, Rollx12]...[, Rollx1M]
    {  :        :        :        -   N x M matrix}
    RollxN1 [, RollxN2]...[, RollxNM]]
{                                                                         }
[MODEL
  CALFA, Calfa, DCa/DFz, DCa/DV, D2Ca/DFz2
  PEAKMU, Peak-Mu, DMup/DFz, DMup/DV
  SLIDEMU, Slide-Mu, DMuS/DFz, DMuS/DV
  PKSLIP, Peak-Slip, DSp/DFz, DSp/DV
  TRAIL, Pneumtc-Trail, DXp/DFz, DXp/DV
  LATRL, Lateral-Stiff, DCy/DFz, DCy/DV
  NOMINAL, Fzo, Vo]
{                                                                         }
```

```
[ALIGN
 [P4, Align-Coeff1, Align-Coeff2, Align-Coeff3, Align-Coeff4]
 [YR, M_Vert_Loads+1, N_Alfas+1
  0.0, Alfa1, ... AlfaN   {Row #1 of (M+1) x (N+1) matrix; 0.0 < Alfa1 < AlfaN ! }
  Vert-Load1, Align-Torque11, Align-Torque12, ... Align-Torque1N
  {     :          :              :                    :    -     (M+1) x (N+1) matrix
  }
  Vert-LoadM, Align-TorqueM1, Align-TorqueM2, ... Align-TorqueMN]]
END
```

<u>{PROTOTYPE SELF-STEER FILE}</u>

```
Title
{3 comment lines (text or blank)}
SLFSTR
ARMXY, Mechanical-Trail, Lateral-Kingpin-Offset
FRICT, Coulomb-Friction                      {torque/axle}
[LINEAR, Aligning-Stiffness, Steer-Damping]  {Phase IV input}
[TABLE, Primary-Aligning-Stiffness, Table-Length {Yaw/Roll input}
  Steer-Angle, Aligning-Torque]              {Table-Length Lines}
[FORCED,                                     { not finalized yet ! }]
END
```

{ Note: All stiffness and damping values are angular (torque-based), per axle. }

{PROTOTYPE STEERING SYSTEM FILE}

```
Title
{3 comment lines  (text or blank)}
STRSTM
KINEM, Steering-Ratio, Mechanical-Trail, Lateral-Offset
STIFF, Steering-Stiffness, Tie-Rod-Stiffness, Wrap-Up-Stiffness
END
```

{PROTOTYPE ANTILOCK FILE}

Title
{3 comment lines (text or blank)}
ALOCK
{A parameter list according to specifications in pp. 257-267, Phase 4 manual, beginning with OPTION1 (p. 257) and terminating with TSMPLE (p. 266)}

{PROTOTYPE STEER-APPLICATION FILE}

```
Title
{3 comment lines (text or blank)}
/ ANGLE, Table1-Length
 Time,  Steer-Angle /                          {Table1-Length lines}
/ LRANGLE, Table2-Length
 Time, Left-Wheel-Angle, Right-Wheel-Angle /   {Table2-Length lines}
/ PATH, Table3-Length
 X-Path, Y-Path                                {Table3-Length lines}
 DRIVER, Driver-Lag, Preview-Interval
 [SWITCH, Closed-Loop-TimeOut, Ramp-Steer-Rate] /        {see note}
END
```

{ Note:
SWITCH line may figure only immediately after a DRIVER line. If no SWITCH line is present, a continuous closed-loop operation is assumed.
Ramp-Steer-Rate is required in *deg/sec*, and is interpreted as an average front-wheel steer-rate if no steering system (ratio) is specified, or as a steering-wheel turning rate if a steering system (ratio) is specified. A negative Ramp-Steer-Rate implies steering to the left direction from the last steer angle where the driver model has left off.}

{PROTOTYPE BRAKE-APPLICATION FILE}

```
Title
{3 comment lines (text or blank)}
BRKAPL, Table-Length
Time, Pressure          {Table-Length lines}
END
```

Example of Actual Suspension-File:

Hendrickson Walking Beam (44K, 60" axle spacing) Make and Model :
Hendrickson RTE.440
Overall tire width : 96."
Filename = ST6T:Su.HknWkBm44
RSUSP, 2
INERT, 2500., 5100. (weight, roll moment of inertia)
LONGL, 60., 50., 0. (", 50/50 stat., 0 dync. transfer)
VERT, 20., 33. (Hcg, Hrc)
TRACK, 72., 13., 38.
ROLL, 30000., .22 (auxiliary K-roll, roll-steer coeff.)
REAR, **ROLL**, 85000., .23 (as above, trailing axle)
END

Example of an Actual Vehicle Index File:

```
RTAC 8 axle Doubles (49t GCW), conf. 2.1, var. 1.00
* Conventional walking-beam tandem-axle tractor.
* Two 27' tandem-axle semi's.  72" single-axle B-dolly.
FileName = ST6T:In.2.1C1.00
VEHICLE, 4
  TRKTR, ST6T:Tr.3ax190wb
STRSTM, ST6T:St.IH12k.Pwr
LOAD, ST6T:SL.Tr4.511.5
FSUSP, ST6T:Su.IH12kFrnt
SPRING, ENVLP, ST6T:Sp.IHref.Frt
WHLBRK, LINEAR, ST6T:Br.StrAxle
RSUSP, 1, ST6T:Su.HknWkBm44
SPRING, ENVLP, ST6T:Sp.HknRte440
HITCH, ST6T:Hi.5thWheel
  SEMI, 2, ST6T:Se.27FtTndm
LOAD, ST6T:SL.Se12t
HITCH, ST6T:Hi.BdollyRef
  DOLLY, 3, ST6T:Do.Convrt1Ax
LOAD, ST6T:SL.Do9t
RSUSP, 1, ST6T:Su.Rc21Bsngl
SLFSTR, ST6T:Ss.CESCHI
HITCH, ST6T:Hi.5thWheel
  SEMI, 4, ST6T:Se.27FtTndm
LOAD, ST6T:SL.Se12t
DEFAULT, RSUSP, 1, ST6T:Su.Rc21B48in
DEFAULT, SPRING, ENVLP, ST6T:Sp.MTC.Reyco
DEFAULT, TIRE, TABLE, ST6T:Ti.XZA11R225
DEFAULT, WHLBRK, LINEAR, ST6T:Br.DualsAxle
END
```

Example of an Actual Simulation-File:

```
FileName = ST6T:Si.2.1C1.000
RTAC 8 axle C-double (49t GCW), conf. 2.1, var. 1.00
Peak Ay = 0.15 G @ 100.0 km/h & Ay Sine T = 3.0 s.
Yaw/Roll simulation - Rapid Path Change
VEHICLE, ST6T:In.2.1C1.00
PROGRAM, P4, -
MEASURE, B
STRAPL, PATH, ST6T:Ap.Path3.0S
VELTIM, 91.134, 8   (100 km/h, 8 sec.)
PRINT, -, 0.1
ERDFILE, =, 0.1
END
```

Example of an Actual Tractor Tare File:

```
FileName = ST6T:Tr.3ax190wb    Last Update: 2/10/86
Tare file for a baseline RTAC tandem-axle tractor.
Type :  Long Conventional;   Wheelbase :  190".
Note : Inertial and torsional parameters are estimated.
TRKTR, 11800        (lbs; total tare weight = 18000 lbs)
TFRAME, 1000000., 5000., 38.        (Kt, Cf, Ht)
RSUSP, 1
1, 190
CGXYZ, 55., 0., 44.
INERT, 26000, 170000, 170000       (in-lbs-sec2)
HITCH, 175., 0., 44.
END
```

Examples of Actual Load-Files:

FileName = ST6T:SL.Se12t
This is a load file for an RTAC tandem-axle semitrailer.
This version specifies suspension ground load
and payload density.
SSPLOAD, 1
1, 26455   (12.0 t)
DENSITY, 34.0   (lbs/ft^3)
END

Filename = ST6T:PL.27F16t, Configuration 2.1, Variations 5.10, 5.11
Payload (16t) file for a 27' semitrailer
This file specifies inertias and c.g. locations
PAYLOAD, 35275 (16t)
INERT, 104215, 824242, 878449
CGXYZ, 137.9, 0, 82.7
END

Example of an Actual Semi Tare File:

```
FileName=ST6T:Se.27FtTndm    Last Update:11/22/85
Tare file for a short RTAC tandem-axle semi.
Type :  27' Van;        Suspension CL to rear end :  54".
Kingpin setting :  24".
SEMI, 5500     (estimated)
KINGPIN, 24
BEDXYZ, 324, 102, 54
RSUSP, 1
1,246
CGXYZ, 165, 0, 69
INERT, 37813, 268334, 268334    (in-lbs-sec2)
HITCH, 300, 0, 34.5
END
```

3.3.1 File Naming Conventions

Naming Conventions for RTAC Simulations Input and Output Files on MTS

All MTS permanent files generated by or for RTAC simulation runs should include in their names basic information (within the unfortunate 12 character constraint) indicating the type of file and its general contents. This is done by making each file name to consist of two distinct, concatenated parts: A **Prefix** (first 2 characters followed by a period) identifying the file type, and a **Body** (characters #4 up to #12) specifying its contents. As there is a discrete number of valid file types, there will be the same discrete number of corresponding valid prefixes which should always be used.

The following **Prefixes** have been defined so far:

File Type	Prefix (characters 1-3)
Simulation-File	Si.
Vehicle-Index	In.
Steer- or Brake-Application	Ap.
Tare-Files:	
Tractor or Truck	Tr.
Semi	Se.
Dolly	Do.
Load-Files:	
Suspension-Load	SL.
Payload	PL.
Hitch	Hi.
Steering-System	St.
Suspension	Su.
Spring	Sp.
Wheel-Brake	Br.
Tire	Ti.
Anti-Lock	AL.
Self-Steer	SS.
ERD output	ER.
Printout	Pr.

3.4 SIMULATIONS

3.4.1 Phase 4

The Phase 4 program is a braking and directional response time-domain mathematical simulation of a truck/tractor, a semitrailer, and up to two full trailers. The vehicles are represented by differential equations derived from Newtonian mechanics that are solved for successive time increments by digital integration.The program is written in a generalized fashion to allow simulation of a large number of vehicle configurations.

3.4.2 Yaw Roll

The Constant Velocity Yaw/Roll Model simulates the turning and rolling behavior of motor vehicles in constant speed maneuvers. The model's particular features are tailored to simulation of trucks and tractor-trailer, accommodating up to four vehicle units. The simulation is particularly versatile in representation of multiple-axle configurations and different types of hitching mechanisms between the vehicle units. It generates time-based output indicating motions of each vehicle and the controlling forces internal to the vehicles.

3.4.3 Static Roll

The Static Roll Model is useful for calculating the rollover threshold of articulated vehicles during steady turning maneuvers. The roll response in a steady turn is computed by repeatedly solving, for small increments of roll angle, a set of equations which describe the static equilibrium of the vehicle in the roll plane.

3.4.4 Simplified Braking

The Simplified Braking Model determines the braking performance of an articulated vehicle assuming that it is making a constant deceleration stop. The simulation will accommodate a vehicle composed of a truck/trailer, a semitrailer, and up to two full trailers. The response to the applied braking forces is described in terms of the longitudinal deceleration and the vertical loads carried by each axle.

3.5 POST PROCESSING PROGRAMS

The Post Processor program is the final step in calculating results in the Simulation Interface. It takes as input the ERD_file created by the run of the computer simulation, and condenses the data into a few values that indicate the important features of the simulation. The program calculates the measures that are indicated in the file runfile, as described in Section 2.2.3.

The RTAC Database is a database containing the results of all the simulation runs made, as specified in the simulation matrix. The Database Updating program adds the new vehicle performance measures calculated by the Post Processor program to the RTAC Database. This database can later be transferred to a spreadsheet package to examine the results of the vehicle simulations.

4.0 INTERACTION OF INTERFACE COMPONENTS

This section discusses how the various components of the simulation interface interact to produce the final results of a simulation.

4.1 RUNSIM MACRO

The macro Runsim is essentially a manager of the programs in the simulation interface. It channels the information flow between programs, and directs the flow of the interface.

Runsim is executed by issuing the command Runsim, and supplying the simulation file to be used. Runsim first creates (or empties if they already exist) all the files that will be used by the programs in the Simulation Interface. It also issues some commands to suppress the output to the screen, so the user is not bombarded by messages that are not necessary.

It then runs the interface program, ST6T:vs.obj, with the simulation file supplied in the Runsim command. The output files from this program are saved in either permanent or temporary files, and named as indicated in the simulation file. Runsim then checks to make sure that the interface program ran correctly, by checking the file inputok. If the program did not run successfully, then Runsim writes a message to this effect on the printfile and immediately exits the macro (skipping the simulation and post processor). It also saves the debug file as a permanent file (according to the naming convention) for later inspection.

The next step in the Simulation Interface is the running of the specified simulation. Runsim branches to the simulation specified in the Simulation file and issues the appropriate command to run that simulation. In this run command, Runsim assigns the output from the interface program as input to the simulation program. It also channels the output from the simulation to the appropriately named printfile and ERDfile.

The final program in the Simulation Interface, the Post Processor, is then run. It takes as input the ERDfile output from the simulation run. The output from this program is stored in the file measures.out, and is appended to the beginning of the printfile by Runsim.

Finally, Runsim performs some cleanup operations. It copies the printfile to the printer, and destroys or empties the files that contain data that are no longer needed. The last task that Runsim performs is the issuing of the commands to re-assign the output to the screen.

4.2 INTERFACE PROGRAM

The Interface program takes as input the Simulation file, which indicates the simulation to be run, the vehicle to be examined, and the vehicle performance measures to be calculated, in addition to other parameters related to the running of the simulation. The output from this program is contained in four files, Destination, Disp.out, Debug.out, and Runfile. All of these files were discussed earlier in section 2.2.1, except for Runfile. Runfile, an input file for the Post Processor, contains one line which indicates the vehicle performance measures that are to be calculated in the Post Processor program.

4.3 SIMULATION PROGRAMS

The simulation programs (Phase 4, Yaw/Roll, Simplified Braking, Static Roll, and the Linear Yaw Plane models) interact with the rest of the Simulation Interface in a very simple way. The interaction occurs through three files, Destination file, Printfile, and ERDfile. Everything the simulation needs as input is contained in the file Destination. The Printfile is not used by the rest of the Simulation Interface, but is used as a means of communicating the results of this process with the user. The ERDfile is used by the Post Processor program to compute the measures that are indicated in the file Runfile (created by the interface program).

4.4 POST PROCESSOR PROGRAM

The Post Processor program takes as input the ERDfile created by the vehicle simulation run. The ERDfile contains a binary record of the output of the simulation run. The Runfile created by the Interface program is also an input file to the Post Processor, it indicates which measures are to be calculated for the simulation run being examined. The output from this program appears in two files, Measures.out and Measure.temp. Measures.out is a text file displaying the results of the vehicle performance measures, Measure.temp is a binary record of these same results, used as input to the Database Updating program.

4.5 DATABASE UPDATING PROGRAM

The Database Updating program takes as input the file Measure.temp, created by the Post Processor program. This program uses the information in this file to update the vehicle performance measures stored in the RTAC Database file. This program is run after each vehicle simulation is completed, maintaining the RTAC Database as a completely up to date record of the results of every simulation run completed in the simulation matrix.

ERD Files

SUMMARY

The Engineering Research Division (ERD) file format can be used for disk or tape files whenever data are organized by channel number and sample number. The file is divided into two sections. The header section includes the information that would normally be included in a log sheet summarizing the data, and can be expanded to include additional information required for specific applications. The data section is similar in concept to a multitrack tape recording. The data can be stored in binary form for maximum efficiency for data processing, or in text form for transferring files between different computers. When the files are stored on reel-to-reel tape, a disk-based directory can be used to improve access to the data.

This manual defines the ERD file layout, and describes some of the software that is available for dealing with ERD files. Utility programs are available on MTS for manipulating the files and plotting the data they contain. In addition, there is a "toolbox" library that can be used to access and change the information in an ERD file when writing new software.

1. INTRODUCTION

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 the mainframe computer at The University of Michigan, both sections are always included in the same file. On the IBM PC, it is usually more convenient to have two separate files having the same base name but with different extensions.

The data section is organized in a form similar to a multitrack tape recorder, with one or more separate channels that are all sampled at identical intervals. The data are stored in binary form when efficiency is important, such as when the ERD files are processed on a single computer system. Alternatively, the data can be stored in text form, to facilitate transporting the files between different computers.

The header section of the file contains the information needed to read the data. This design allows a data processing program to first read the header information that maps out the file, and then read the data. Thus, a program that can deal with one ERD file can probably 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.

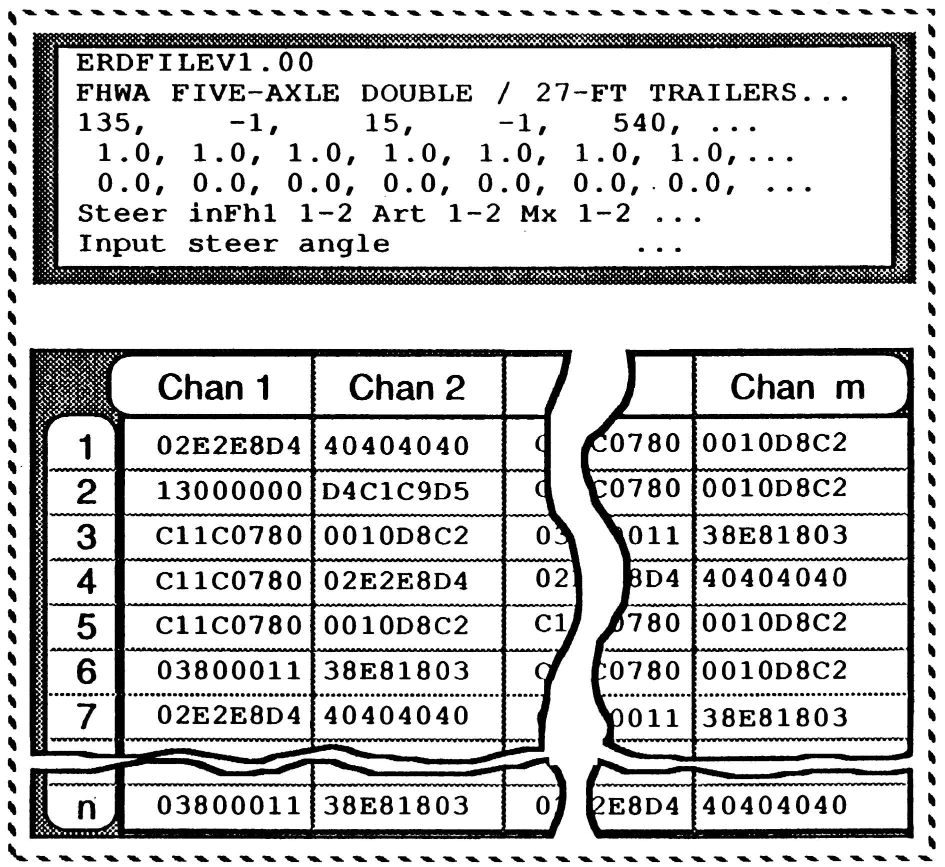
The data are stored in columns and rows as shown in Figure 1. The columns correspond to separate channels, and the rows correspond to samples taken of all of the channels at a particular instant.

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. THE LAYOUT OF AN ERD FILE

This section defines the structure of the ERD file. Examples involving the reading and writing of ERD files are provided later in section 5.

ERD File Structure



```
ERDFILEV1.00
FHWA FIVE-AXLE DOUBLE / 27-FT TRAILERS...
135,     -1,     15,     -1,    540, ...
 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0,...
 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, ...
Steer inFh1 1-2 Art 1-2 Mx 1-2 ...
Input steer angle              ...
```

	Chan 1	Chan 2			Chan m
1	02E2E8D4	40404040	C	C0780	0010D8C2
2	13000000	D4C1C9D5	C	C0780	0010D8C2
3	C11C0780	0010D8C2	03	011	38E81803
4	C11C0780	02E2E8D4	02	8D4	40404040
5	C11C0780	0010D8C2	C1	780	0010D8C2
6	03800011	38E81803	C	C0780	0010D8C2
7	02E2E8D4	40404040		0011	38E81803
n	03800011	38E81803	0	2E8D4	40404040

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

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]

Table 1 summarizes the required lines in the header of an ERD file. The first line identifies the file as an ERD file and the next seven lines contain values for parameters in a particular order. Additional lines can optionally be included that contain information specific to various applications.

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 unit names are defined as containing 8 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, consider a case where the ERD file contains three channels of data, and the units of the variables of the different channels are meters, millimeters, and degrees. Line 8, which gives the names of the units for each channels, might be:

`'m.......mm......deg'`

(For clarity in this discussion, the line of text has been enclosed in single quote marks, and spaces have been shown as periods. In the actual file, the quote marks would be omitted and the period would be replaced with spaces.) The last item can also be padded (and usually will be when the files are created automatically), but this is not absolutely necessary.

All integer and floating-point numbers must be separated with commas. When writing, the numbers should be written in a reasonably compact form, so that programs reading the data will not miss any of the numbers due to too many spaces in the line. Decimal points are optional for floating point numbers such as *Step* on line 3, but integer numbers (such as all of the other numbers on line 3) should never have decimal points because they could cause read errors with some Fortran programs.

In addition to the required eight lines shown in Table 1, any number of additional lines can be added to the header so long as the value of *Nxlines* in line 3 is correct. The additional lines are included because it is sometimes necessary to add specific types of information to the ERD files in some applications which are not relevant for other applications. The optional lines are included as the means for the ERD file format to grow and evolve politely. 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

[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. Conversion between ASCII and EBCDIC is usually accomplished transparently whenever files are transferred between computers. (When conversion is not automatic, utility programs are always available to make the conversion.) On microcomputers, the header of an ERD file will always be an ASCII file. On some mainframes, it will be an EBCDIC file. Throughout the remainder of this manual ASCII/EBCDIC files are referred to simply as "text."

Table 1. Summary of Records in an ERD File Header

Line No.	*Description*
1	*ERDFILEV1.00* — identifies file as having ERD format
2	**Title** (80 characters, left-justified and padded with blanks)
3	Nchannels, Nsamples, NxLines, NRecs, NbytesRec, KeyNumType, Step, KeyOption — use commas to separate numbers **Nchannels** = Number of data channels **Nsamples** = Number of samples. (If unknown, use -1.) The total number of numbers in the data portion of the file is: Nchannels × Nsamples. **Nxlines** = Number of additional lines after line #8. **Nrecs** = Number of records of data. (If unknown, use –1.) **NbytesRec** = Number of bytes/record in binary data **or** number of samples/record for text data. for binary data, should be chosen such that each record begins with channel 1: that is, NbytesRec = K × Nchannels × BytePerNum, where K is an integer and BytePerNum is the number of bytes/data value (2 for integer, 4 for floating-point, etc.). For text data, the number of numbers per line is NbytesRec × Nchannels . **KeyNumType** = Indicates how the data are stored. 0=2-byte integer (binary), 1=4-byte floating point (binary), 2=proposed for 8-byte floating point (binary), 3=proposed for 8-byte complex (binary), 4=proposed for 16-byte complex (binary), 5=Formatted floating-point (text) The format is (Nchannels)E13.6 unless a FORMAT optional keyword is used to specify a different format.. **Step** = sample interval (time step,frequency step,count) **KeyOption** = optional key used in some data processing applications.
4	**Gains** (scale factors) for each channel, separated by commas. Use 1.0 (for each channel) if data values are already scaled correctly.
5	**Offsets** for each channel, separated by commas. 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+ NxLines)	**Optional records.** Each record should begin with a 8-character keyword (GENNAME, TESTID, etc.) These records will contain 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 **NxLines.**
9+NxLines	**First** data record.
8+NxLines +NbinRecs	**Last** data record.

use for that information, then it will recognize the keyword. If not, the rest of the line is not processed and no harm is done.

An example of an optional line is:

```
XUNITS..sec
```

(As in the earlier example, periods are shown instead of the spaces that would be used in the actual file.) This line identifies the units of the independent variable implicit in the structure of the ERD file. (That is, the numbers in the data portion of the file are sampled at constant intervals of an implicit independent variable, measured in units of 'sec'.) In this case, the eight-character keyword is 'XUNITS '. This keyword is associated with a name of the units used, sec. This piece of information might be of no use for a program that reduces the time histories to a single summary index, and such a program would just skip over this line when it does not recognize the keyword 'XUNITS '. On the other hand, when the file is read by the ERD plotter described in section 4, this information is used to help label the x axis when time histories are selected.

So far, all of the key words that have been used fall into eight categories of information. They include four data types: 8-character names, 32-character names, 80-character names, and numbers. There can be one piece of information that applies to the entire file (four of the categories), or one piece per channel (these are the other four categories.) Table 2 lists some of the keywords that have been used to date.

2.2 The Data

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 data points 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 2-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 (The University of Michigan computer system): 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.

Table 2. Summary of Keywords Recognized by the Toolbox

Keyword Name	*Description*	*No. of Values*	*Variable Type*
AXLETRAK	Dimensions for each axle, in inches.	n	number
AXLEWT	Weights of each axle assembly, in pounds.	n	number
FSTAXLES	The number of the first axle on the units in a multiple vehicle combination.	n	number
FORMAT	Fortran FORMAT statement to be used to read data in ERD file when KEYNUM = 5. (ex: (4F10.4))	1	char*32
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. This keyword can be used repeatedly to add information if the data is modified by post-processing	1	char*80
HITCHKEY	Codes that give the hitch types for a vehicle simulation	n	number
NAXLES	Total number of axles in a vehicle.	1	number
PROFINST	Name of profilometer instrument used to measure data	1	char*32
ROLLCNTR	Heights of roll centers above ground, in inches.	n	number
ROLLHT	Vertical distances between the roll centers of each axle and the c.g. of the corresponding sprung mass. The units are inches.	n	number
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	number
SPRUNGWT	Weights of sprung masses, in pounds.	n	number
STEERIN	Name of steering input to vehicle (e.g., trap steer) .	1	char*32
TESTID	Number used to identify a test.	1	number
TRUCKSIM	Name of the simulation model.	1	char*32
WHOBLAME	Person to contact with questions about the data.	1	char*32

Table 2. Summary of Keywords Recognized by the Toolbox — continued

Keyword Name	*Description*	*No. of Values*	*Variable Type*
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	number
XUNITS	Units of independent variable (e.g., sec).	1	char*8

Binary data are not readable by humans. When viewed with a text editor, binary data will be unintelligible. It is always necessary to have a program read the binary data into an array in memory, and then display the data by printing or plotting.

Different computers use different methods to represent numbers in binary form. Different software packages on the same computer may use incompatible representations for storing floating point numbers. However, all computers can read numbers when they are written out in text form. 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 should include the optional 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; however, there are penalties for using text representations of numbers. 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 ten times greater for numbers in text form rather than binary. On smaller computers such as the IBM PC, 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 requires 4 bytes / number in binary, and 13 bytes / number as text. On microcomputers with floppy disk drives, disk space can be a serious limitation. Binary files can be three times longer than text files.

3. TAPES

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

3.1 Initialization

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 used in the ERD files should be binary (integer 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 32767. (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.)

3.2 Disk Directory

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.

Applications

From the experience so far, the disk directories have proven to be almost essential for automated processing, and so convenient for interactive processing that they might almost 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 plotting data from ERD files. 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 then 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 and can be corrected.

Format

The format of the disk directory is quite simple, and is shown in Table 3. 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 next (8+Nxlines) 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 exists that will create a disk directory that references only selected files on the 9-track tape. It is described in Section 6.

Table 3. Summary of Records in Disk Directory

Sector	*Line (s)*	*Description*
	1	"LONGDIR" followed by tape name. (Note: Quotes should not be entered in the directory file. 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 (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	ERDFILEV1.00
	6 to 3 + **NL**	Rest of header for some tape file. (ex: line 6 => Title for 8th tape file...)
2	4 + **NL**	Number of another file on tape and its name
	5 + **NL**	ERDFILEV1.00 (Start of header info for next tape file)
	3 + 2 • **NL**	Last line available for header info
		•
		•
		•
NS	4 + (NS - 1) • **NL**	Number and name for last tape file
	3 + **NL** • **NS**	last line of directory file

4. 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-handling 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. The plotter has the 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
- 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

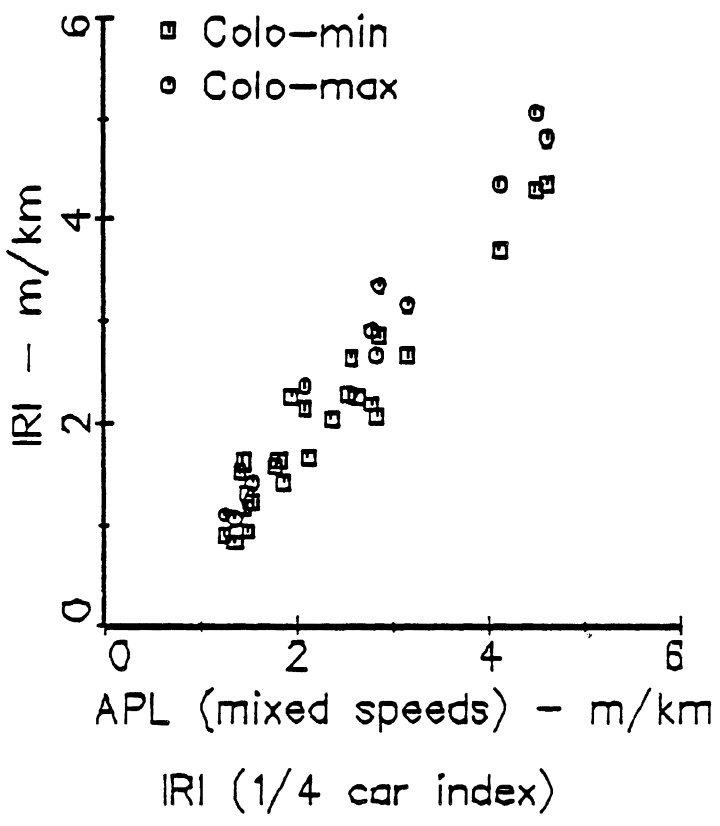


Figure 2

This plot shows summary data in a scatter plot. The ERD file contained all of the summary values, with each "channel" in the file corresponding to a different instrument, and each "sample" corresponding to a different test site.

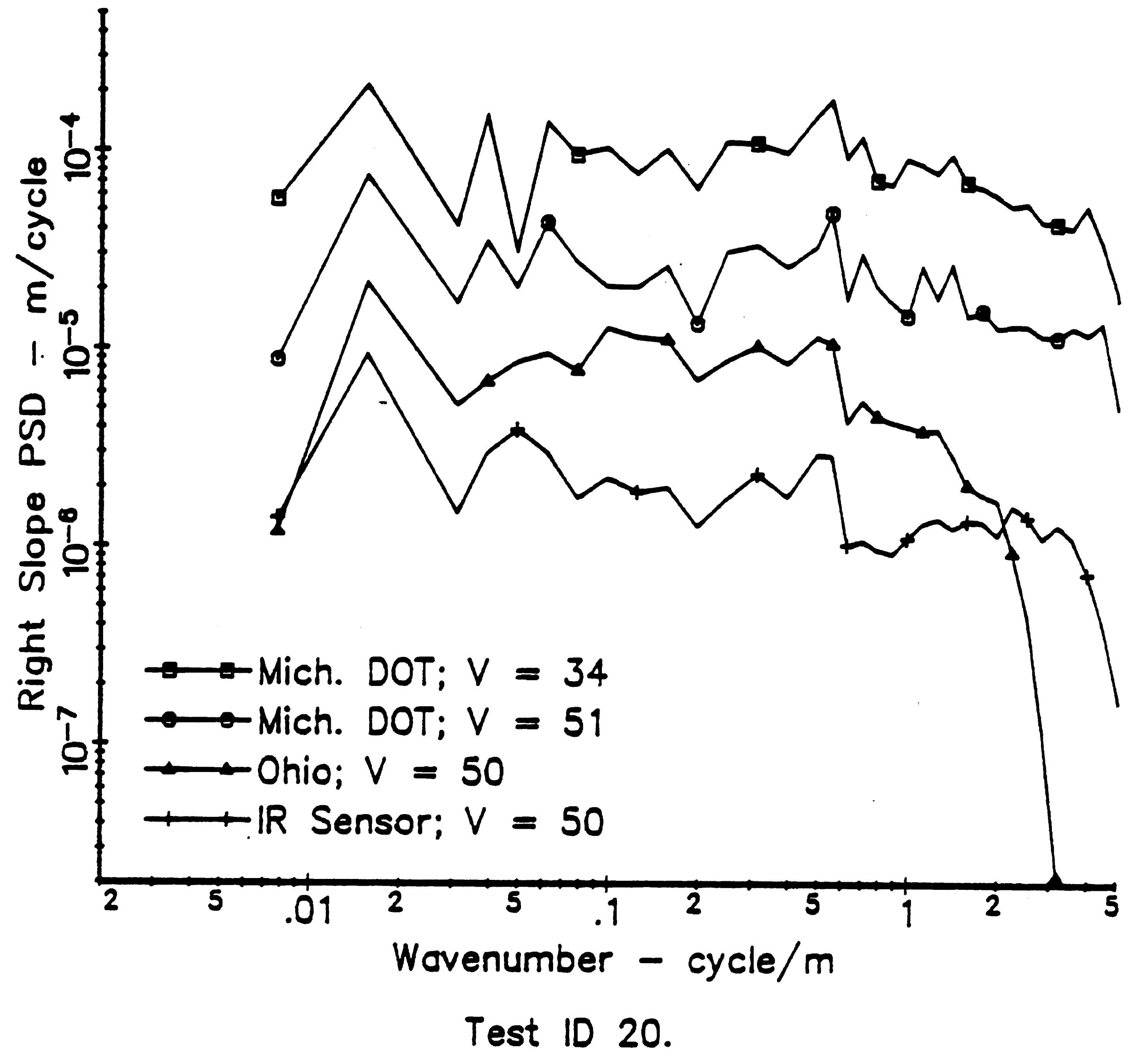
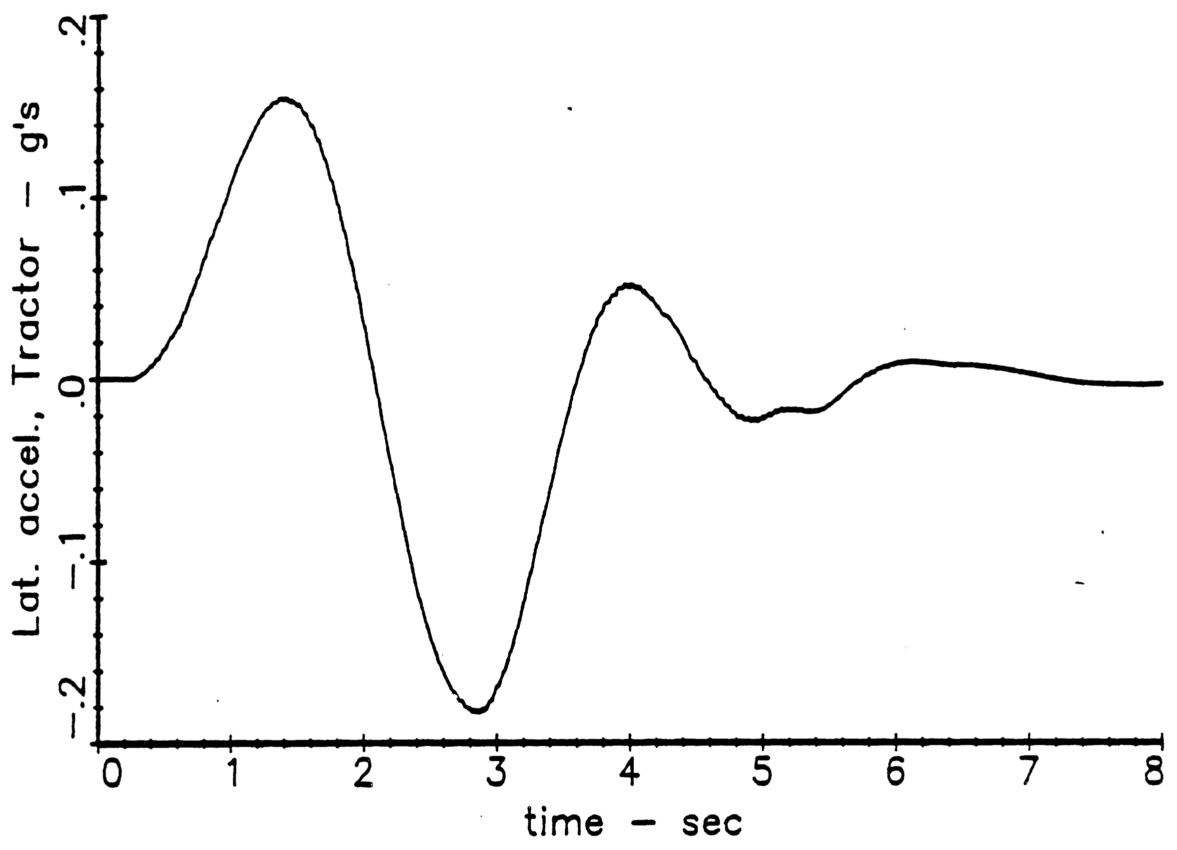


Figure 3

This plot shows data from 5 ERD files plotted on log axes. The scaling and range of the x axis were set manually, while the range of the y axis was determined automatically and the scaling was set at 1-decade/inch. The plotter offset the lines so that the individual plots could be distinguished more easily. The plotter chose the "long name" for the vertical axis because all channels had the same name. It combined several names to label the symbols, so that each symbol would have a unique name. (The labels used there were "instrument name" and "test speed.") The plotter chose the test ID to label the entire figure, because all of the ERD files had the same ID in common (and nothing else).



RTAC 8 axle C-train Doubles (49t/108k GCW), conf. 2.1, var. 1.00

Figure 4

This simple plot shows one variable from a simulation model as a function of time. The scaling was automatic and the axes were placed out of the plotting area. The horizontal and vertical axes were labelled with "long names" associated with the selected channels.

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.

4.1 Running the Plotter

The plotter and the supporting libraries live 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.
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 svnl:pl.erdp.o+svnl:sc.lib+svnl: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.)

DEBUG is more expensive than RUN, so 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 4.2 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.

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 stq4:erdp
#$DEBUG
STQ4:NEW.ERDP.O+SS0K:PLOT.O+ss0k:erd.tool.o+SS0K:SIG.TOOL.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 the *Instructions for Use* (below) 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.

4.2 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 still be valid.

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 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

The following instructions are separated into three sections, covering the three levels of control. 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. When a blank line is entered, by just hitting the Return key, this action is shown (as «Return»), to indicate that the computer paused until an entry was made.

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. Also, there is better error checking when a disk directory is used. (Instructions for making a disk directory file are included in the ERD File documentation.) 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 c8116j 9tp *t* vol=prof9 'prof9'
#c8116j 9tp *t* vol=prof9 'prof9'
#*T* (C8116J): Mounted on T904
```

This is the minimum effort needed to mount a labelled tape. The rack number is the number at the top of the receipt for the tape (a computer card), and in this case it is C8116J. 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 svnl:pl.erdp.o+svnl:sc.lib.o+svnl: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 the PLOT documentation.)

```
 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. ERDP will not ask for a plotting range for cross plots involving any choice other than channel 0 for the source of x values, even if zooming is requested here.

```
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»
```

The option to filter the signals is not available when scatter plots are selected.

```
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 - 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, }
2,3,4,5,8
```

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 to the ERDP program which choices are preferred. ERDP will not even consider using a label unless it is allowed here. When several choices are valid, such as 2,3,4,5, and 8 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 8 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.

```
***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 you 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, and even when it is not, the same channel will often be used for the x axis in repeated plots. Therefore, 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 that 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., `I-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 invalid name is entered, the problem is quickly detected and no errors will occur. If an invalid tape name is entered and there is no disk directory, then the entire tape will be searched (taking several minutes sometimes), and possibly 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 ERP 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
```

A CTRL-C was entered, Since no files have been opened for this plot, ERDP takes this to mean that we are through, and returns control to DEBUG. We could start again, possibly to use a different plot format.

```
+mts
#
```

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

4.3 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.

It 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. 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 4.2. Because the setup file has no structure to speak of, this is really the only simple way to generate a setup file for ERDP. 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. Setup files are so easy to create that there will probably never be any conversion utilities written.

Tape Directory Files

The layout of the directory files is described in section 3.2. Plans are underway to begin using a second type of directory file, and therefore the first line of a directory file must indicate which type it is. 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 by-pass the normal control dealing with setup files and channel names. Instead, the template customizes the plotter so that only file names are needed. When processing a tape, the entire tape can be processed automatically using a template.

The template file structure is shown in Table 4. The plotter reads the channel names from the template file as if they were types 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.

Programming Details—Subroutines Used in ERDP

The ERDP program uses a number of special subroutines, contained in several libraries, which were all named in the example run commands shown earlier in section 4.2. The plotting subroutines and the "user-friendly" input routines are contained in the library svn1:pl.lib.o, which is documented elsewhere. The routines specific to the plotter (and contained in the file svn1:pl.erdp.o) are described below. Most of the routines in the toolbox, described in section 5.2, were written after the plotter and are not used. The few exceptions are also listed below.

Table 4. Layout of Plot Template Files.

Line	*Variable*
1	*Plot Template for ERDP, Feb 1986* — this line identifies file to ERDP
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(2,1), 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(NPLOTS,ND(NPLOTS)), chx(NPLOTS,ND(NPLOTS)) — Name of y, x channels for last data set of last plot

DERIV — Takes the derivative of a signal.

GETERD — Get the name of a valid ERD file, which may be on disk or tape, and open the file.

HILOF — Filters a signal using a moving average.

LRSLOP — Finds the slope of a signal using a least-squares-fit linear regression.

OPNSET — Get the name of a valid ERDP setup file and open it.

OPNTMP — Get the name of a valid ERDP template file and open it.

POSNTF — positions a tape and a corresponding disk directory to the beginning of the next ERD file.

RDERD — Reads selected channels from an ERD file.

RDDIR — Get the names of tapes and ERD disk directories, open the directory files, read the names of the ERD files from the directories, and close the directory files.

RDSET — Read the plotting format from an ERDP setup file and close the file.

RDTMP — Read a template file (or series thereof) and close it.

READHD — Read all of the header information from an ERD file.

UNIQUE — Count the number of unique labels in a character array.

WHICHC — Find which channel was selected from an ERD file.

5. 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 at UMTRI to conveniently access the information in ERD files from programs written in Fortran 77.

Examples in Fortran

The following examples illustrate how ERD files can be read and written when the toolbox described in subsection 5.2 is not available.

The ERD Header

Use Fortran read/write statements with regular FORMAT statements to create the lines of text that make up the header, treating the tape (or file) as a Fortran logical unit. For example, the following line of code writes line #4 with gains for each channel:

```
WRITE (2, '(32(E13.6,'','''))') (GAIN (I), I=1, NCHAN)
```

Note that a comma is inserted between numbers to allow for reading the numerical data using a program that does not use the same FORMAT.

Names of the channels and other labels are also read and written using conventional Fortran READ and WRITE statements. For example, the following code could be used to read the short, long, and unit names (lines 6, 7, and 8 in the header) from a file attached to unit #1:

```
CHARACTER*32 LONGNM(20)
CHARACTER*8 UNITS(20), SHORTN(20)
      •
      •
      •
READ (1, '(20A8)') (SHORTN (J), J=1, NCHAN)
READ (1, '(20A32)') (LONGNM (J), J=1, NCHAN)
READ (1, '(20A8)') (UNITNM (J), J=1, NCHAN)
```

Binary Data

To read and write binary data at UMTRI, it is necessary to use the MTS subroutines READ and WRITE. 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(20,20000)
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.

6. UTILITY PROGRAMS

There are several stand-alone programs on the SVN1 account 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  .FP      .FUNCP   .SPLIT
```

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

As input, this program has 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 ERD 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 — Creates 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 3.

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

.SPLIT — Splits 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 with the same number of channels and the same basic information in the headers. The copies will have the data stored either in binary form or as text, as requested by the user. The output files will be contiguous.

ERD FILE OUTPUTS FROM THE VEHICLE SIMULATIONS

The Phase 4 and yaw/roll models have recently been extended to produce ERD files containing the time histories of the simulation variables. An ERD file includes numerical data in the same form as a multi-channel tape recorder, and also includes a great deal of labelling information. This document describes the labels that are used, including new labels defined specifically for these models. It also includes some example listings from the Phase 4 model and the yaw/roll model.

The minimal ERD file contains three labels for each channel: a short name eight characters long, a longer name containing up to 32 characters, and a name for the units that is eight characters long. The ERD files from the simulations include an extra two labels for each channel that are identified by the keywords GENNAME and RIGIBODY. 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 - 200 channels. When dealing with so many channels, standard naming conventions are helpful. Most of the time, the short names will be of primary interest, because they will be used to select channels in post-processing programs.

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 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 rearmost trailer. Dollies are treated as sprung masses. Therefore, a doubles combination will have four sprung masses (tractor, 1st semitrailer, dolly, 2nd semitrailer). A triples combination will have six sprung masses (tractor, 1st semitrailer, 1st dolly, 2nd semitrailer, 2nd dolly, 3d semitrailer).

Rigid-Body Names

The variables in the ERD file can be associated with the various rigid bodies that comprise the vehicle model. In the ERD files, each channel is given a label that identifies its associated rigid body. The rigid-body names are an extension to the ERD file format, and are stored in an optional line that begins with the letters *RIGIBODY*. There are five classes of names, each with a different naming convention.

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) .... *Semitrailer*
	(4 or more sprung masses) .... *1st Semitrailer*
Sprung mass #3	(4 sprung masses) .... *Dolly*
	(6 sprung masses) .... *1st Dolly*
Sprung mass #4	.... *2nd Semitrailer*
Sprung mass #5	.... *2nd Dolly*
Sprung mass #6	.... *3d Semitrailer*

(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

Each channel in an ERD file can be given a general name, to aid in labelling plots of several channels. Each name is up to 32 characters in length, and is stored on a line in the ERD file that begins with the keyword GENNAME. 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 Accleration, Slip Angle,* and *Y Position.* Below is a list of all General Names currently used in the Phase 4 and yaw/roll simulations.

Length	*Spring Deflection*	*X Position*	*Y Position*
	Z 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*	

Units

The names of the units are limited to eight characters. The spellings used in the ERD files are the same as used below

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)	

Long Names

Each channel in the ERD file is also given a unique name that can be up to 32 characters long. 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 Semitrailer*; *X Position, Axle 3*; *Articulation Angle, Hitch 1-2.* The convention used for the long names is not critical for most applications, since channels are typically identified using the short names. 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.* When necessary, the word "Axle" is abbreviated as "Ax" to stay within the 32 character length limitation. Also, the GENNAME will be abbreviated as necessary; e.g., *Longitudinal Slip, R Side, Ax 12, Long. Accel., 2nd Semitrailer.* The two input variable names are *Input Steer Angle* and *Brake Treadle Pressure.*

Short Names

Each channel in the ERD file is given a unique name that is limited to eight characters in length. 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 six 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 preceeding blank if it will fit within the allowable eight character length. Therefore, a maximum of six characters are left to describe the variable itself. The names for the 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 requires up to two characters at the end of the name for the axle number, leaving a maximum of four 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 eight character length (e.g., *L Fz 2, R Fz 2, R Alph 5, R Alph11.*).

4. Hitches: These names will always end with three 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 **RIGIBODY** and **GENNAME**, described earlier, the following optional keywords are included in the ERD files generated by the simulation programs.

AXLETRAK$n,t_1,t_2, \dots t_n$	n is the number of axle track values that will follow, and is equal to the number of axles. t_1 - t_n are the track dimensions for each axle, in inches.
AXLEWT $n,w_1,w_2, \dots w_n$	n is the number of axle weight values that will follow, and is equal to the number of axles. w_1 - w_n are the weights of each axle assembly, in pounds.
FSTAXLES $n, i_1,i_2,\dots i_n$	n is the number of integers that will follow, and is also equal to the number of trailers. i_1 is the number of the first axle on the second unit; i_2 is the number of the first axle on the third unit, and so on. These data provide the only way to identify the specific sprung mass that is associated with a particular axle. For example, a doubles combination vehicle might have a line that reads `FSTAXLES 3,3,5,6` This indicates: that there are 4 sprung masses (3 trailers plus the lead vehicle unit); that axles 1 and 2 are on the lead unit; that axles 3 and 4 are on the first semitrailer; that axle 5 is on the dolly; and that axles numbered 6 or higher are on the second semitrailer. Dollies are counted as trailers for this keyword, even when the ERD file is generated by the Phase 4 model.
HISTORY *message*	*message* is a string of text, up to 80 characters in length, that tells what model generated the run. It also gives the time and date of the run.
HITCHKEY$n,i_1,i_2, \dots i_n$	n is the number of integers that will follow, and is also equal to the number of hitches. i_1 -i_n are codes, each with a value between 1 and 6 that gives the hitch type for the associated hitch. (1=5th wheel, 2=Inverted 5th wheel, 3=Compensating 5th wheel, 4=Turntable, 5=A-dolly, 6=B-Dolly)
NAXLES n,	n is the total number of axles in the vehicle being simulated.
ROLLCNTR$n,h_1,h_2, \dots h_n$	n is the number of height values that will follow, and is equal to the number of axles. h_1 - h_n are the heights of the roll centers above the ground, in inches.
ROLLHT $n,h_1,h_2, \dots h_n$	n is the number of height values that will follow, and is equal to the number of axles. h_1 - h_n are the vertical distances between the roll centers of each axle and the c.g. of

	the corresponding sprung mass. The units are inches. A positive value means the c.g. of the sprung mass is above the roll center for that axle. When the sprung mass is a dolly, a value of 0.0 is written if the ERD file is generated by the Phase 4 program.
SPEEDMPH*v*,	*v* is the starting speed for the run, with units of mi/h.
SPRUNGWT$n,w_1,w_2, \dots w_n$	*n* is the number of weights that will follow, and is equal to the number of sprung masses. w_1 - w_n are the weights of each sprung mass, in pounds. When the ERD file is generated by the Phase 4 program, weights of 0.0 are used for any dollies.
TRUCKSIM*name*	*name* is the name of the simulation model, e.g., *Phase 4 12-85, Yaw/Roll 12-85*
XLABEL time	The independent variable is named "time."
XUNITS sec	The name for the units of time is "sec."

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  Fhl 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  Fhl 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  Fhl 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                          }
  •
  •
  •
```

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        }
```

PLOT: A PLOTTING PACKAGE FOR ENGINEERING APPLICATIONS

PLOT is the name of a one-size-fits-all subroutine aimed at engineering applications involving line and scatter plots using linear and log plotting scales. It generates plots using the primitive graphics routines in the *Plotsys library at MTS. Using the PLTEND library, the plots can be viewed immediately on Tektronix terminals (or Macs running the Versaterm program) and they can also be plotted using an x-y plotter. This document describes how to set the parameters passed to the PLOT subroutine to control the various scaling and formatting options that are available. It also describes the library of plotting subroutines that are used by PLOT.

OVERVIEW

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 idea is that you provide the data, and PLOT does the rest. 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, and in addition, a number of additional library subroutines to add capabilities beyond those available in *Plotsys.

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

- **FP** is for plotting data contained in text files (FP stands for File Plotter). FP was written by Sayers in 1982±, and can be used for text files having arbitrary layout. With this program, you type in the scaling options, labels for the axes, and the format used in the text file. It helps to know a little about how the PLOT subroutine works when using FP. An example FP run is included in this documentation.
- **ERDP** will plot data from ERD files. ERDP is the second generation of the FP program, and is much easier to use because all of the labels used for plotting are contained in the ERD file. It was written by Sayers in 1985, and has its own documentation.

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.

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 ith 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 jth point, XYDATA(2,j) = y value for jth 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 ith data set. The data points can be shown in four different ways:

NSYMB(i) = 0	*Line Plot*: x-y points in ith 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 ith 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 ith 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 ith 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

There are lots of ways the PLOT program can scale and size plots. 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 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 non-zero 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 $\neq$ 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 $\geq$ 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 $\geq$ 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 $\neq$ 0

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

[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.

|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.

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 1 summarizes the ways that the display of text is determined.

Table 1. 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 Formats	Formatting of numbers on linear and log scales is set by the PLOT subroutine. Numbers are shown conventionally for $.01 \le \|x\| < 10000$. 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.

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) $\neq$ 0) and when KEY(3) $\neq$ 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.

The Subroutine Call...

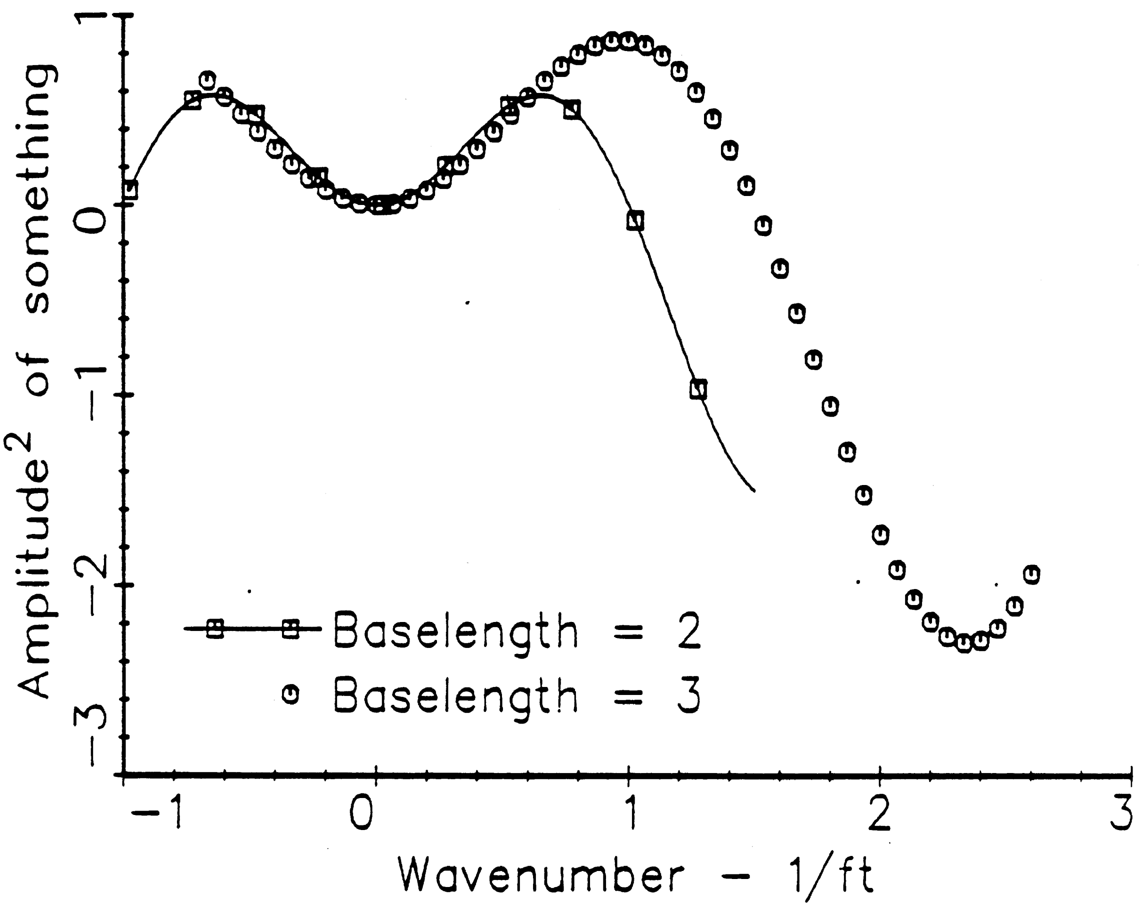
```
CALL PLOT (XYDATA, NPTS, NDSETS, NSYMB, XL, YL, TITLE, DESC, RANGE,
           KEY)
```

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 PLOT subroutine assumes that the variables are stored as x1,y1, x2,y2, x3, ... (Thus it is usually convenient to dimension the array as 2 $\times$ ntot, where ntot $\geq$ 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 1. 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



Example use of the PLOT subroutine

Figure 1

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/
*
      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
```

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      cccccccccccccccccccc
      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 fp.o+plot.o+*ig+*plotsys
```

fp.o is the program being run. Plot.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     , 0.0      , 0.0     ,      5.00,
  3.00, 0.5000E+00, 0.5000E+00, 0.150}  «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). 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 2 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.

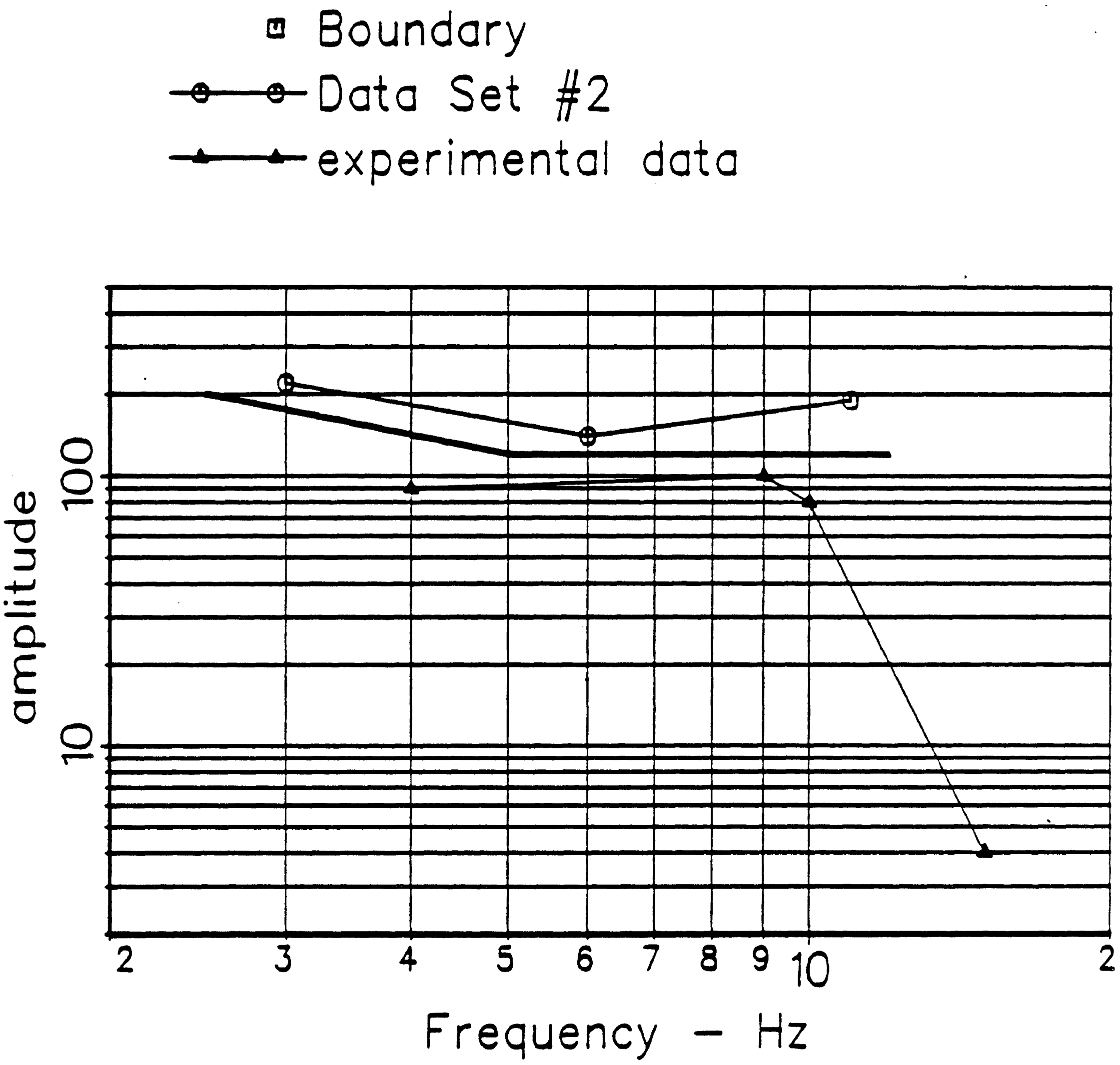
```
 PDS: PLOT DESCRIPTION GENERATION BEGINS.
 Blow-up, Redraw, Plot or Continue? c
```

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
```



Demo plot made with the FP program

Figure 2

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 2 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. The file names for the source and object code on MTS are given in italics (later). 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.

AUTOLG (DMAX, DMIN, MAX, MIN) *s/o*

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) *s/o*

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

Table 2. 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 from paper coordinates to engineering units.

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 from engineering units to paper coordinates.

SCLDWN (X, XNORM, XDOWN) — round a number down to next multiple of 1, 2, or 5.

SCLUP (X, XNORM, XUP) — round a number up to next multiple of 1, 2, or 5.

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 marks for a linear axis.

X1EQX2 (X1, X2) — don't let X1 = X2.

AUTOTK (AXLEN, MAX, MIN, TICK, HT) *s/o*

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) *s/o*

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) *s/o*

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 coordinatewhere 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) *s/o*

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 viewpoot (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) *s/o*

Write a number into a string for use as a label on an axis. Fixed format is used for $.1 \leq x < 100{,}000$. 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)*s/o*

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 counter-clockwise 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*4	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) *s/o*

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) *s/o*

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 counter-clockwise 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) *s/o*

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) *s/o*

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 $\neq$ XMIN, then the subroutine will constrain the search for YMAX and YMIN to those points where XMIN $\leq x \leq$ 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 $\neq$ 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) *s/o*

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) *s/o*

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

→ X	real*4	number to be re-scaled (not changed by subroutine). This number must be ≥ 0.
← XNORM	real*4	normalized input, re-scaled 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) *s/o*

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

→ X	real*4	number to be re-scaled (not changed by subroutine). This number must be ≥ 0.
← XNORM	real*4	normalized input, re-scaled 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) *s/o*

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(θ), where θ = 0 for vertical tick and θ=90° for horizontal tick.
→ SINA	real*4	sin(θ), where θ = 0 for vertical tick and θ=90° for horizontal tick.
→ SIZE	real*4	1/2 the length of the tick line (inches).

TIKRND (MIN, MAX, TICK) *s/o*

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) *s/o*

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) *s/o*

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

Interfacing Adaptive Control Features in the Phase 4 Model

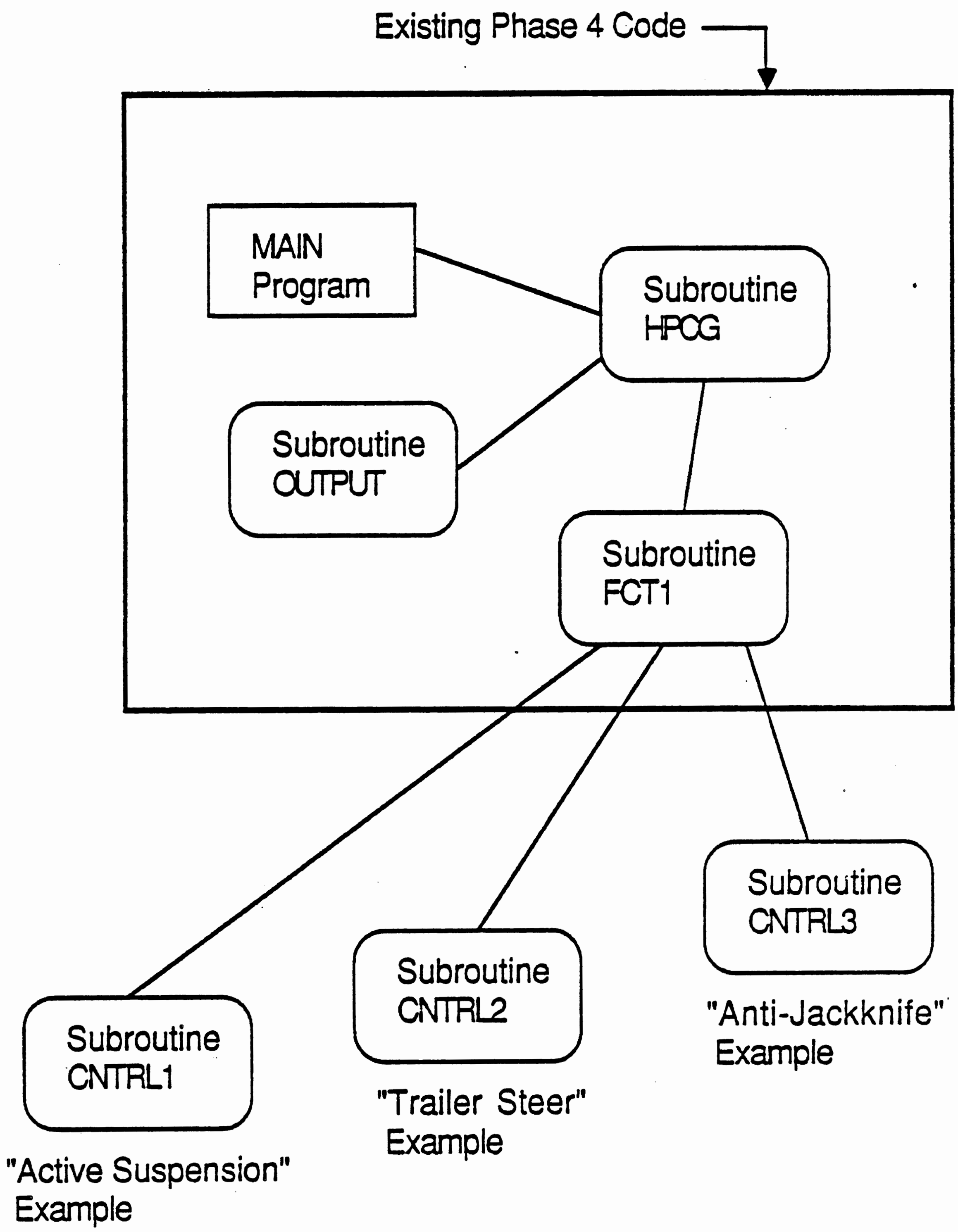
Interfacing Adaptive Control Features in the Phase 4 Model

The material in this section describes how to modify the existing Phase 4 program for adding adaptive control features to the model. Three examples are used to illustrate the "programming mechanics" needed to implement: a) an active suspension feature , b) steering of semitrailer wheels, and c) brake pressure modulation responding to a tractor-semitrailer "jackknife" condition during braking.

The usual method for adding specific features, not already present in the Phase 4 program, is to add the desired code in a subroutine, and then call that subroutine from an appropriate location within the existing Phase 4 program. The trick, of course, is to know from *where* in the Phase 4 program to call the user-written subroutine and *which* Phase 4 variables to use or modify within the user subroutine. Since the Phase 4 program was originally intended to represent what seemed at the time a good variety of common vehicle configurations (present in the fleet in the mid-to-late 1970's), no provision for modification or extension to other types of vehicle configurations was designed into the original code. Consequently, changes or additions to the code to study some of the more unusual vehicle configurations increasingly present in the fleet today usually requires some familiarity with the program structure and variables in order to implement additional or new features. Short of producing an extensive document detailing the purpose of each program variable and an accompanying flow chart to assist anyone interested in performing arbitrary changes to the model (a major project in itself), the material presented here is intended to be introductory and tutorial in scope. The examples selected for this purpose cover three basic adaptive control areas seen as likely of interest to most users, namely — modification of suspension forces, steering of wheels other than the conventional front axle wheels, and brake pressure modulation based upon some vehicle response condition.

The following figure illustrates how each of the example subroutines, identified as Subroutine CNTRL1, CNTRL2, AND CNTRL3, interface with the existing Phase 4 code. Each of these user-written subroutines are called from the Phase 4 subroutine FCT1. Subroutine FCT1 in the Phase 4 code is called within the integration loop to evaluate the right-hand side of the differential equations defining the model. Consequently, any calculations related to suspension or tire forces, for example, need to be evaluated ultimately within this subroutine. A recommended method for minimizing the amount of direct alteration of existing Phase 4 code when new features need to be added, is to simply

Interfacing the Adaptive Control Examples to Phase 4



place a call to a user-written subroutine from the FCT1 subroutine, and perform the new or additional calculations within the external user subroutine.

The following three examples show, first, how each of the three subroutines are called from the Phase 4 subroutine FCT1 (new code shown as bold-faced type in the Phase 4 listings). The additional code (bold-faced) which calls each new control subroutine is located within that part of the FCT1 subroutine associated with the particular control feature. For example, subroutine CNTRL1 (active suspension example) is called from a location in subroutine FCT1 following the standard suspension force calculations.

Following these listings which show the required modifications to the Phase 4 code in subroutine FCT1, figures and accompanying code are then presented for each of the control examples. The figures illustrate the basic control concept being implemented; the accompanying listings show the FORTRAN code defining each example control subroutine called by the Phase 4 program. Example time-history results for each control concept are also shown following each subroutine listing using the post-processor plotter described in the previous section.

Example of Call to Subroutine CNTRL1 from existing Phase 4 code.

Phase 4 Code - Subroutine FCT1

```
              .
              .
              .

C
C   Z VELOCITY OF SPRUNG MASS
C
            SMSD = UU * A(IVEH,1,3) + VV * A(IVEH,2,3) + WW * A(IVEH,
     1      3,3)
            IF (IVEH .GT. 1) GO TO 140
            IF (MVEH .EQ. 1) GO TO 140
            IF (JSUS .EQ. 1) GO TO 140
            SMS = SMS + YS * (PHI2C - PHIBAR(1,1))
            VV = VV - (FRZ(1,2,KAX) - HFRAME) * PBAR(2,1) + FRZ(1,2,
     1      KAX) * PBAR(1,1)
            WW = WW + YS * (PBAR(2,1) - PBAR(1,1))
  140       CONTINUE
C
C   CALL  SUSPENSION ROUTINE
C
            CALL TANDIN(IVEH, JSUS, KAX, LSIDE, SMS, SMSD, SIGN)
C
C   CALL FOR SPRING FORCES
C
            CALL LINE(IVEH, JSUS, KAX, LSIDE, SF, K, C, CF, X)
  150     CONTINUE
  160 CONTINUE
C
C   ACTIVE  SUSPENSION  EXAMPLE  /  ROLL  STABILIZATION
C
C   VEHICLE  UNIT  LOOP
C
        DO 165  IVEH=1,MVEH
C
C   SUSPENSION  LOOP
C
         DO 164  JSUS=1,2
C
C
C   SKIP  FRONT  OF  SEMI
C
         JKK=1
          IF(IVEH.EQ.2.AND.JSUS.EQ.1)  GO  TO  164
C
C   TANDEM  AXLES
C
          IF(KEY(IVEH,JSUS).GT.0)  JKK=2
C
C   TRACTOR  FRONT  AXLE
C
          IF(IVEH.EQ.1.AND.JSUS.EQ.1)  JKK=1
```

```
C
C   AXLE LOOP
C
               DO 163 KAX=1,JKK
C
C    SIDE-TO-SIDE  LOOP
C
                  DO 162 LSIDE=1,2
C
C    CALL  ACTIVE  SUSPENSION  SUBROUTINE  FOR  EACH  WHEEL  LOCATION:
C
                          CALL  CNTRL1(IVEH,JSUS,KAX,LSIDE,SF,X)
  162           CONTINUE
  163        CONTINUE
  164     CONTINUE
   165  CONTINUE
C
C
C
C   KINEMATIC RELATIONSHIPS BETWEEN THE STATE VARIABLES
C
      DO 180 J = 1, MVEH
        DO 170 I = 1, 3
          IV = (J - 1) * 28 + I
          DERY(IV) = 0.
          DO 170 IXX = 1, 3
C
C   THESE RELATE VELOCITIES IN INERTIAL X,Y,Z TO U,V,W
C
            DERY(IV) = UBAR(J,IXX) * A(J,IXX,I) + DERY(IV)
  170   CONTINUE
C
C   THESE RELATE RATES OF CHANGE OF PHI,THETA,PSI TO P,Q,R
C
        DERY((J - 1)*28 + 6) = PBAR(J,3) + PBAR(J,2) * SIN(PHIBAR(J,1))
        DERY((J - 1)*28 + 4) = PBAR(J,1) + PBAR(J,3) * TAN(PHIBAR(J,2))
     1  + PBAR(J,2) * PHIBAR(J,1) * PHIBAR(J,2)
        DERY((J - 1)*28 + 5) = PBAR(J,2) - PBAR(J,3) * SIN(PHIBAR(J,1))
  180 CONTINUE
C
C   NOW WE COMPUTE THE POSITIONS AND VELOCITY OF THE ENDS OF THE DOLLY
C   FOR USE IN DOLLY FORCE CALCULAION.  A IS AT PINTLE HOOK,  B IS AT
C   TURNTABLE.
C
      IF (MVEH .LT. 3) GO TO 240
      DO 230 IVEH = 3, MVEH
        WW = 0.
        VV = 1.
        TONGLE(IVEH) = 0.
        IVM = IVEH - 1
            .
            .
            .
```

Phase 4 Code - Subroutine FCT1 continued

Examples of Calls to Subroutines CNTRL2 & CNTRL3 from existing Phase 4 code.

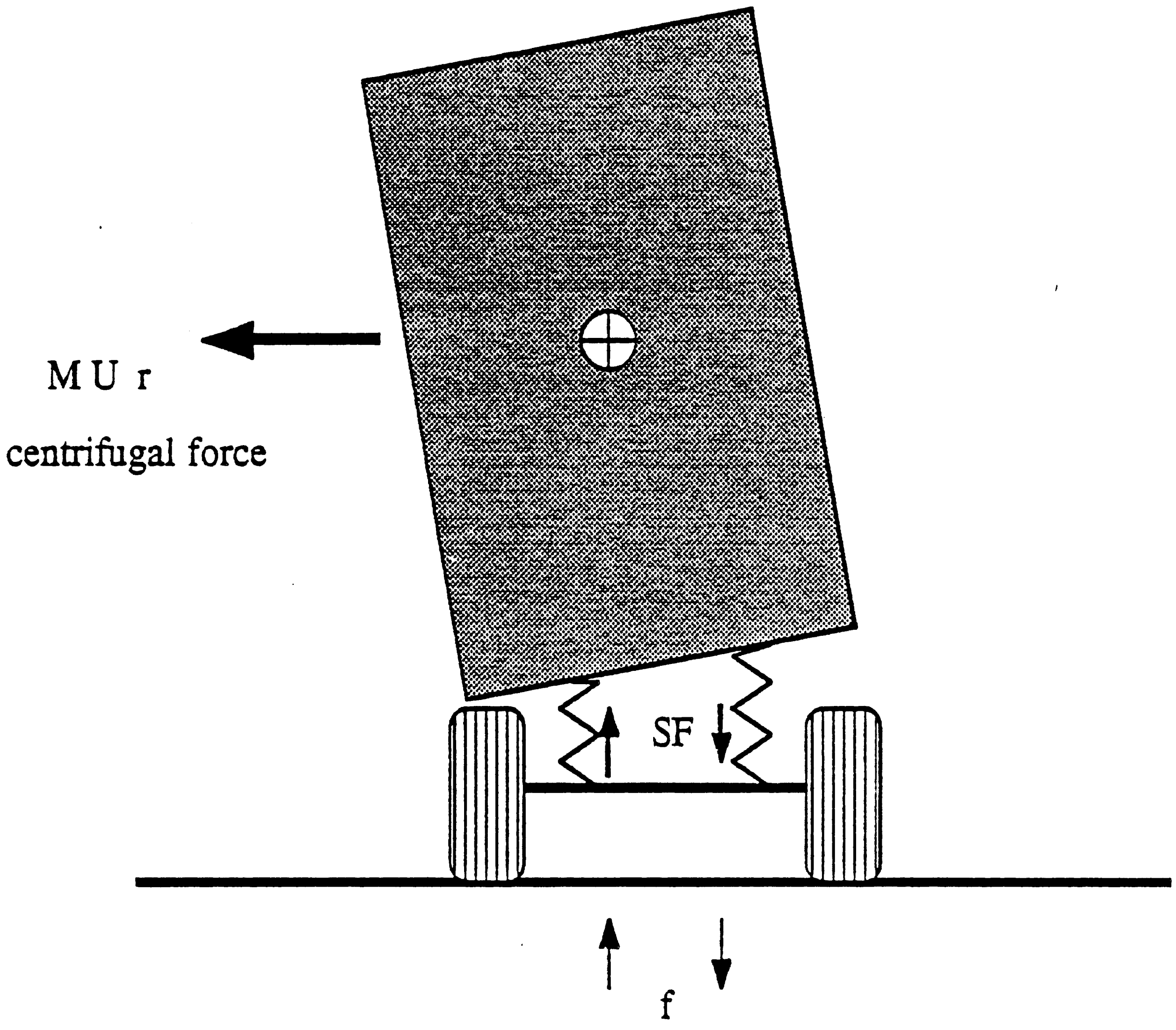
Phase 4 Code - Subroutine FCT1

```
           .
           .
           .
C
      DO 290 IVEH = 1, MVEH
C
C   FSUM AND TSUM ARE FOR EQUATIONS OF MOTION. FX,FY,TZ ARE FOR
C   CONSTRAINT FORCE CALCULATIONS
C
        FX(IVEH) = GVWM(IVEH)*G*DZDXI(IVEH)
        FY(IVEH) = GVWM(IVEH)*G*DZDYI(IVEH)
        TZ(IVEH) = 0.
        DO 250 JV = 1, 3
          FSUM(IVEH,JV) = 0.
          TSUM(IVEH,JV) = 0.
  250   CONTINUE
        DO 290 JSUS = 1, 2
C
C   NO FRONT TIRES ON THE SEMI-TRAILER
C
          IF (IVEH .EQ. 2 .AND. JSUS .EQ. 1) GO TO 290
          JKK = 1
          IF (KEY(IVEH,JSUS) .GT. 0) JKK = 2
          IF (IVEH .EQ. 1 .AND. JSUS .EQ. 1) JKK = 1
          DO 280 KAX = 1, JKK
            DO 270 LSIDE = 1, 2
C
C   ROLL STEER CALCULATION
C
              IF (JSUS .EQ. 2) DELT(IVEH,JSUS,KAX,LSIDE) = 0.
        IF(INDPKY(IVEH,JSUS,KAX).EQ.1) GO TO 255
              DELT(IVEH,JSUS,KAX,LSIDE) = DELT(IVEH,JSUS,KAX,LSIDE) +
     1        RST(IVEH,JSUS,KAX) * (PHIBAR(IVEH,1) - THETAX(IVEH,JSUS,
     2        KAX))
      GO TO 258
  255 DELT(IVEH,JSUS,KAX,1)=DELT(IVEH,JSUS,KAX,1)+RST(IVEH,JSUS,KAX)
     1*ZAXLE(IVEH,JSUS,KAX)
      DELT(IVEH,JSUS,KAX,2)=DELT(IVEH,JSUS,KAX,2)+RST(IVEH,JSUS,KAX)
     1*THETAX(IVEH,JSUS,KAX)
C
C   TRAILER STEER OPTION EXAMPLE --   1985/6 MVMA PROJECT
C
  258 CONTINUE
          CALL  CNTRL2(IVEH,JSUS,KAX,LSIDE)
C
              CALL BRAKE2(IVEH, JSUS, KAX, LSIDE, XXS, X)
C
C   BRAKE RELEASE DUE TO HIGH SPEED JACKKNIFE EXAMPLE
C   1985/6 MVMA PROJECT
C
C                 CALL  CNTRL3(IVEH,JSUS,KAX,LSIDE)
C
              DO 260 IXY = 1, 2
C
```

```
C   CALCULATE ALL TIRE FORCES
C
  260         CALL TIRE(IVEH, JSUS, KAX, LSIDE, IXY, T, DT, SRS, XXS,
     1             DERY, TALIGN, X)
              FXI(IVEH,JSUS,KAX,LSIDE) = FXW(IVEH,JSUS,KAX,LSIDE) * COS(
     1        DELT(IVEH,JSUS,KAX,LSIDE)) - FYW(IVEH,JSUS,KAX,LSIDE) *
     2        SIN(DELT(IVEH,JSUS,KAX,LSIDE))
              FYI(IVEH,JSUS,KAX,LSIDE) = FXW(IVEH,JSUS,KAX,LSIDE) * SIN(
     1        DELT(IVEH,JSUS,KAX,LSIDE)) + FYW(IVEH,JSUS,KAX,LSIDE) *
     2        COS(DELT(IVEH,JSUS,KAX,LSIDE))
C
C   FX,FY, AND TZ ARE USED HERE TO SUM THE FORCES FOR USE IN
C   CONSTRAINT CALCULATIONS
C
      XROAD = XBAR(1,1)
        IF (IROAD .EQ. 0) GO TO 268
        DZDX1= DZDX0
        DZDY1= DZDY0
        CPSI(IVEH) = COS(PHIBAR(IVEH,3))
        SPSI(IVEH) = SIN(PHIBAR(IVEH,3))
        IF (IROAD .GE. 0) GO TO 267
            .
            .
            .
```

Phase 4 Code - Subroutine FCT1 continued

Active Suspension Example



Under steady turning conditions:

$$\boxed{f = \underbrace{m\,U\,r}_{} \cdot \text{Gain}}$$

"centrifugal force" @ axle • Gain Factor

f is the active suspension force component

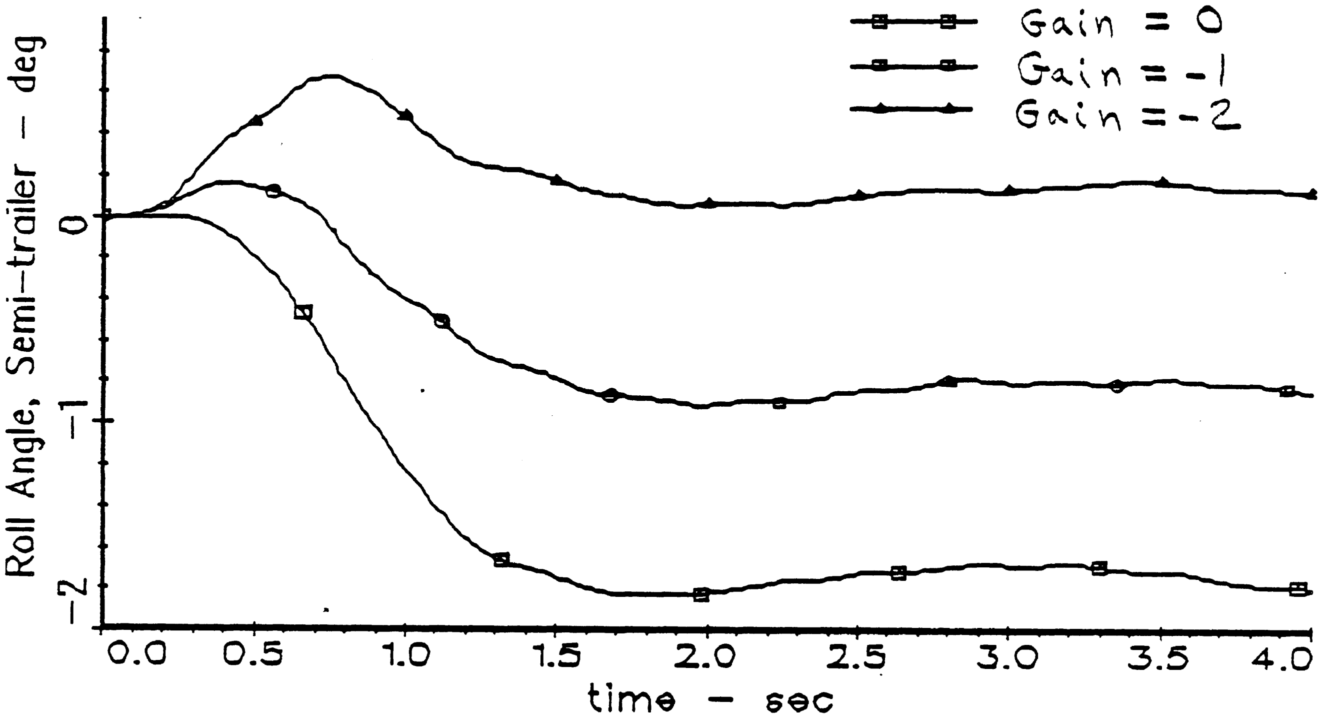
SF is the conventional spring force

```
C
C  EXAMPLE ADAPTIVE CONTROL SUBROUTINE -  Active Suspension Example
C                                         /  Roll Stabilization
C  1985/6 MVMA PROJECT
C
        SUBROUTINE  CNTRL1(IV,IS,IA,ILR,SFF,T)
C
C  ARGUMENT LIST:
C
C  IV IS VEHICLE MASS UNIT (1 TO 4)
C  IS IS UNIT SUSPENSION (1 OR 2)
C  IA IS SUSPENSION AXLE NUMBER (1 OR 2)
C  ILR IS LEFT OR RIGHT SIDE (1 OR 2)
C  SFF IS RETURNED SUSPENSION FORCE MODIFIED BY ACTIVE FORCE
C  T IS TIME
C
      REAL NS
C
C  COMMON BLOCKS FROM MAIN PHASE 4 PROGRAM TO PASS VARIABLES/PARAMETERS:
C
      COMMON /FCTOUT/ XBAR(4,3), PHIBAR(4,3), UBAR(4,3), PBAR(4,3)
      COMMON /TURN/ NOTURN(2,2), TURNX(50), TURNY(50), DELT(4,2,2,2)
      COMMON /STATIC/ NS(4,2,2), FT(4), SF(4,2,2,2)

      DIMENSION GAIN(4,2,2,2), SFF(4,2,2,2), TLAST(4,2,2,2)
C
C  INITIALIZE GAIN FACTORS & PARAMETERS
      DATA GAIN / 32*0.0/
      DATA GRAV /32.2/
      DATA TLAST / 32*-0.1/

C
C  ONLY ENTER ONCE PER TIME STEP
C
      IF(T.LE.TLAST(IV,IS,IA,ILR)) RETURN
      IF(T.GT.0.) GO TO 100
C
C
C  SET GAIN FACTORS AT ACTIVE SUSPENSION LOCATIONS ON LEFT SIDE
C  (WOULD NORMALLY BE READ FROM A FILE)
C
      GAIN(1,2,1,1) = -2.0
      GAIN(1,2,2,1) = -2.0
      GAIN(2,2,1,1) = -2.0
      GAIN(2,2,2,1) = -2.0
C
C  RIGHT SIDE GAINS EQUAL TO LEFT & SIGN CHANGED
C
      DO 40 I=1,4
        DO 30 J=1,2
          DO 20 K=1,2
            GAIN(I,J,K,2) = -GAIN(I,J,K,1)
   20     CONTINUE
   30   CONTINUE
   40 CONTINUE
```

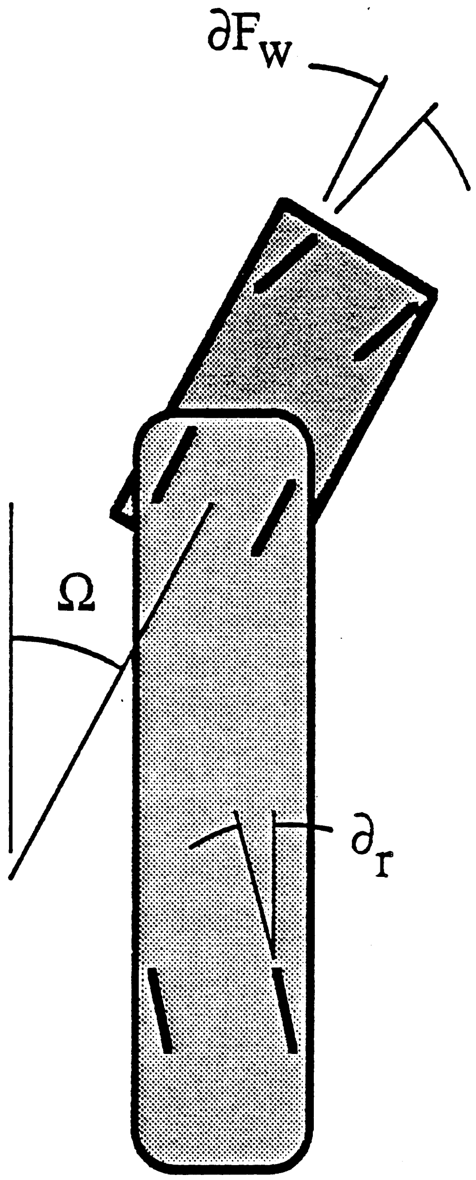
```
C
C  CALCULATE & ADD ACTIVE SUSPENSION FORCE COMPONENT TO PRESENT
C  SPRING FORCE SF.
C
  100 SFF(IV,IS,IA,ILR) = SF(IV,IS,IA,ILR) + GAIN(IV,IS,IA,ILR)*
     1  UBAR(IV,1)*PBAR(IV,3)/GRAV*NS(IV,IS,IA)
C
C  RETURN SUM OF CURRENT SUSPENSION FORCE & ACTIVE FORCE COMPONENT AS SFF:
C
      RETURN
      END
```



5—AXLE TRACTOR—SEMI; 60 mph; Steady Turning Example

Semitrailer Rear Wheel Steering Example

(low speed steering condition shown)

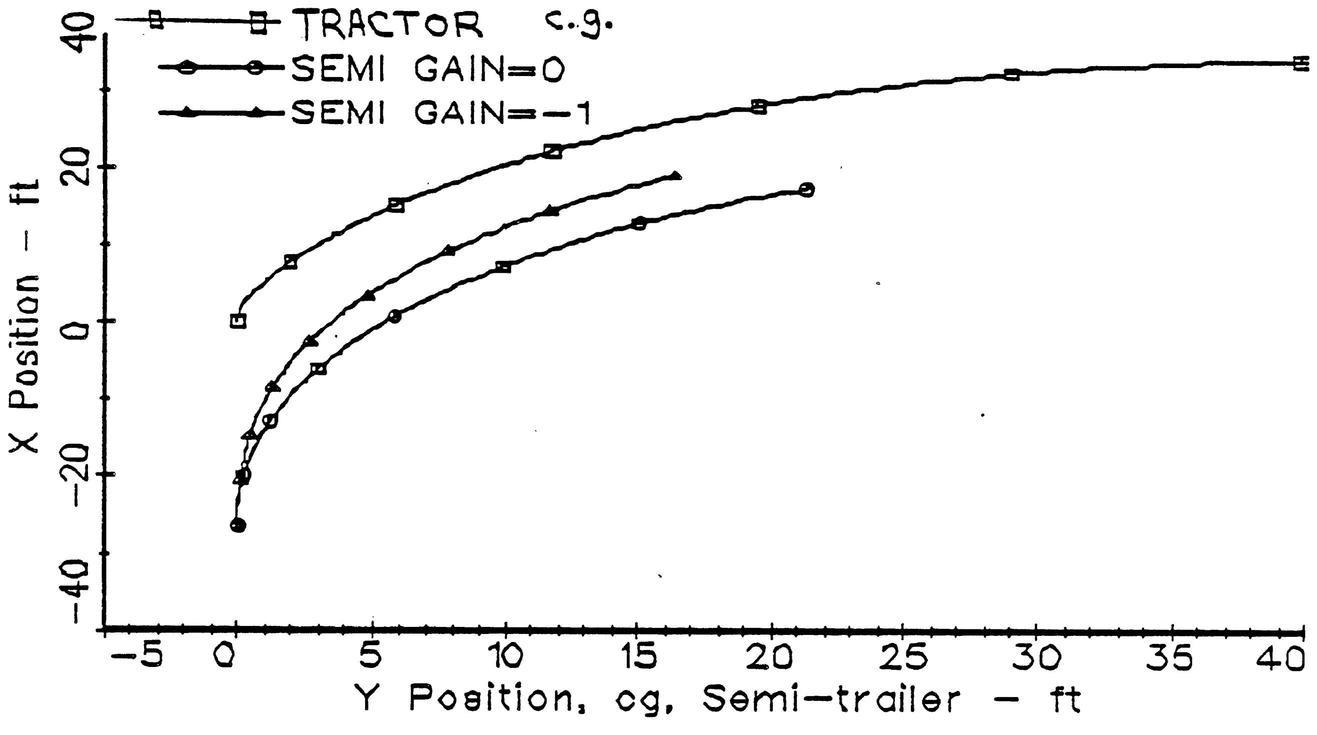


$$\partial_r = \text{Gain} \cdot \Omega$$

$$\text{Gain} = (V - c) / c$$

V is vehicle speed
c is a normalizing constant (e.g. 40 mph)

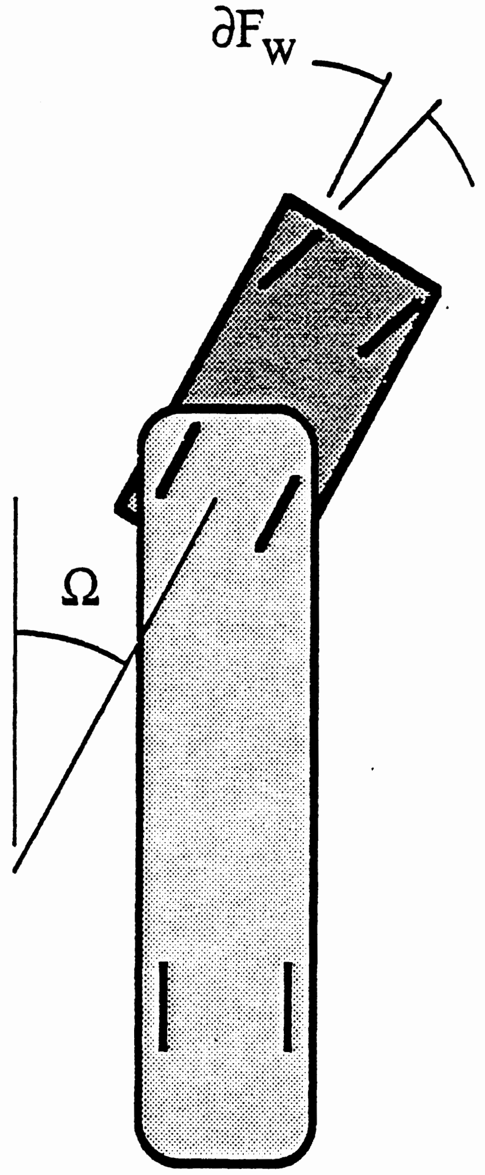
```
C
C  EXAMPLE ADAPTIVE CONTROL SUBROUTINE - Semitrailer Rear-Wheel Steering Example
C  1985/6 MVMA PROJECT
C
         SUBROUTINE  CNTRL2(IVEH,JSUS,KAX,LSIDE)
C
C  ARGUMENT LIST:
C
C  IVEH IS VEHICLE MASS UNIT (1 TO 4)
C  JSUS IS UNIT SUSPENSION (1 OR 2)
C  KAX IS SUSPENSION AXLE NUMBER (1 OR 2)
C  LSIDE IS LEFT OR RIGHT SIDE (1 OR 2)
C
C  COMMON BLOCKS FROM MAIN PHASE 4 PROGRAM TO PASS VARIABLES/PARAMETERS:
C
     COMMON /FCTOUT/ XBAR(4,3), PHIBAR(4,3), UBAR(4,3), PBAR(4,3)
     COMMON /TURN/ NOTURN(2,2), TURNX(50), TURNY(50), DELT(4,2,2,2)
C
C  INITIALIZE NORMALIZING FACTOR
     DATA W00 / 58.0 /
C
C
C  SELECT ONLY SEMITRAILER UNIT
C
     IF(IVEH.NE.2) RETURN
C
C  SELECT REAR SUSPENSION ONLY
C
     IF(JSUS.NE.2) RETURN
C
C  SET GAIN AS FUNCTION OF SPEED:
C
C  (FOR A RIGHT TURN, POSITIVE GAIN VALUES STEER THE SEMI WHEELS TO THE RIGHT).
C  UBAR(2,1) IS THE FORWARD SPEED OF THE SEMITRAILER (FT/SEC).
C  W00 IS A NORMALIZING PARAMETER SELECTED TO PRODUCE CONVENTIONAL (NON)
C  STEERING AT 58 FT/SEC (40 MPH), A GAIN VALUE OF -1 AT LOW (ZERO) SPEED TO
C  OFFSET LOW SPEED OFF-TRACKING TENDANCIES, AND A
C  POSITIVE GAIN AT SPEEDS ABOVE 40 MPH TO OFFSET HIGH SPEED OFF-TRACKING
C  TENDANCIES.
C
      GAIN = (UBAR(2,1) - W00) / W00
C
C  STEER SEMITRAILER WHEELS IN PROPORTION TO ARTICULATION ANGLE
C  & ADD TO CURRENT (ORDINARILY ZERO) VALUE:
C
       DELT(IVEH,JSUS,KAX,LSIDE) = GAIN * (PHIBAR(1,3) - PHIBAR(2,3))
    1 + DELT(IVEH,JSUS,KAX,LSIDE)
C
C  RETURN MODIFIED SEMITRAILER STEER ANGLE, DELT
C
     RETURN
     END
```



5-AXLE TRACTOR-SEMI; Low Speed Steady Turning Example
(+ traction)

Anti-Jackknife Control Example

(high speed braking)



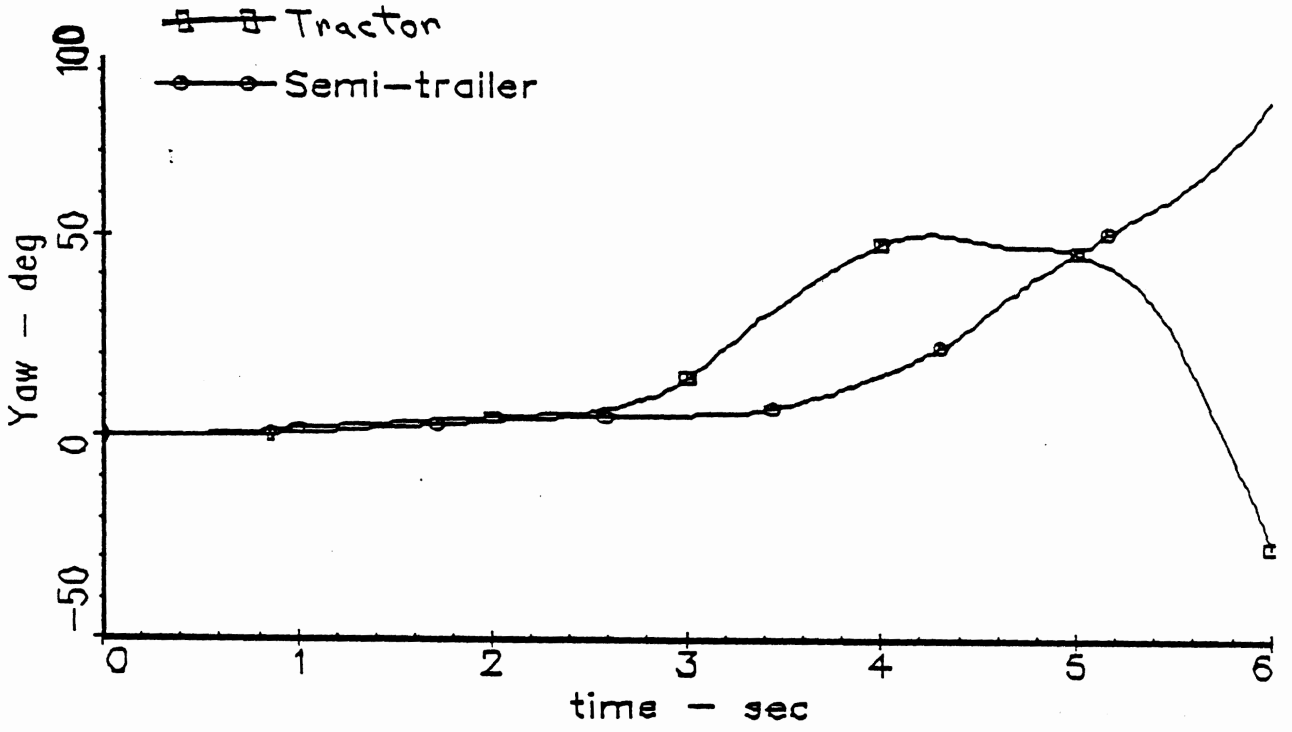
For speeds greater than 40 mph

and

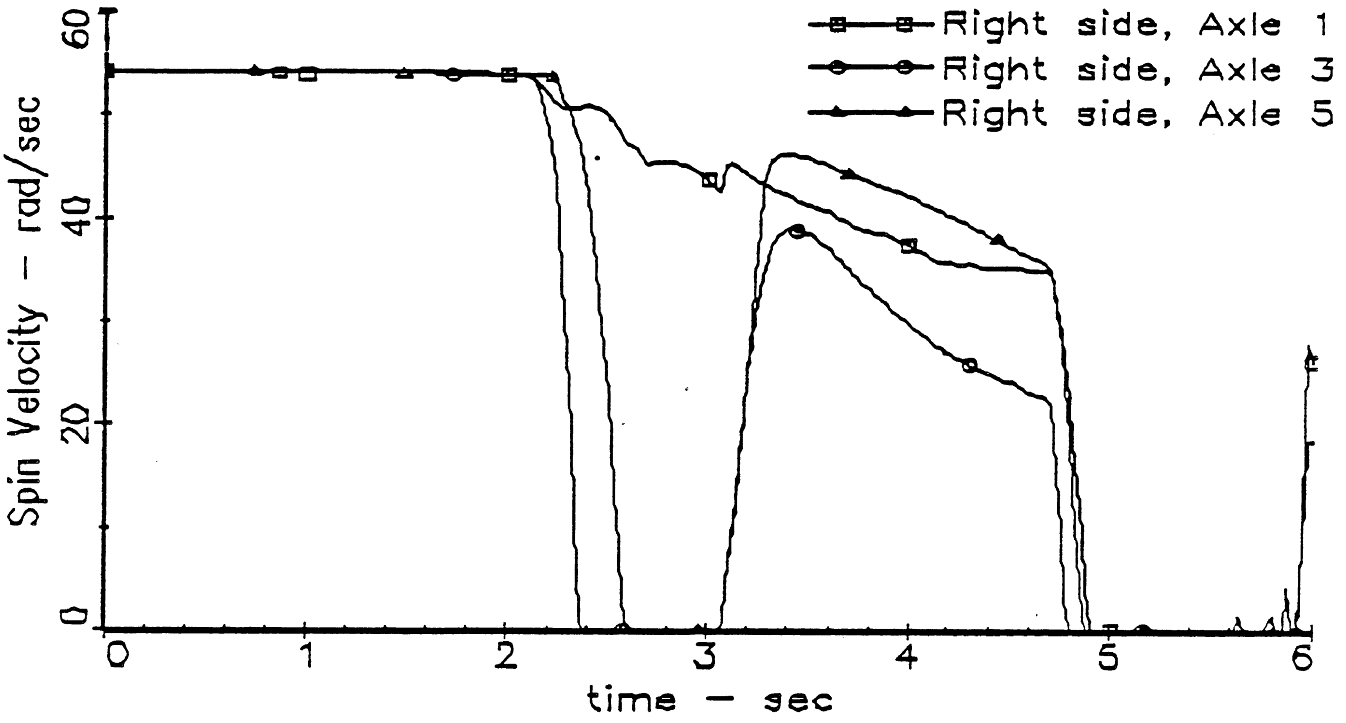
Ω greater than 0.2 radians:

$\boxed{\text{Release the Brakes}}$

```
C
C  EXAMPLE ADAPTIVE CONTROL SUBROUTINE - Anti-Jackknife Control  (Release of
C     Brake Pressure for Large Articulatiuon Angles at High Speed)
C  1985/6 MVMA PROJECT
C
         SUBROUTINE  CNTRL3(IVEH,JSUS,KAX,LSIDE)
C
C  ARGUMENT LIST:
C
C  IVEH IS VEHICLE MASS UNIT (1 TO 4)
C  JSUS IS UNIT SUSPENSION (1 OR 2)
C  KAX IS SUSPENSION AXLE NUMBER (1 OR 2)
C  LSIDE IS LEFT OR RIGHT SIDE (1 OR 2)
C
C  COMMON BLOCKS FROM MAIN PHASE 4 PROGRAM TO PASS VARIABLES/PARAMETERS:
C
      COMMON /FCTOUT/ XBAR(4,3), PHIBAR(4,3), UBAR(4,3), PBAR(4,3)
      COMMON /BOUT/ P(4,2,2,2), P0(4,2,2,2), TLST(4,2,2,2), T(4,2,2,2),
     1       PTRD
C
C
C  RELEASE BRAKE PRESSURE FOR ARTICULATION ANGLES GREATER THAN
C  0.2 RADIANS AT SPEEDS OVER 40 MPH.
C
C  SPEED CHECK:
C  DO NOTHING FOR SPEEDS LESS THAN 58.6 FT/SEC (40 MPH).
C
      IF(UBAR(2,1) .LE. 58.6) RETURN
C
C  ARTICULATION ANGLE CHECK:
C  DO NOTHING FOR ARTICULATION ANGLES LESS THAN 0.2 RADIANS.
C
      IF(ABS(PHIBAR(1,3) - PHIBAR(2,3)) .LE. 0.2) RETURN
C
C  OTHERWISE, SET BRAKE PRESSURE, P, AND TREADLE PRESSURE, PTRD, TO ZERO:
C
      P(IVEH,JSUS,KAX,LSIDE)  =  0.0
      P0(IVEH,JSUS,KAX,LSIDE)  =  0.0
      PTRD  =  0.0
C
C  RETURN MODIFIED (ZERO) BRAKE LINE & TREADLE PRESSURE IN 'BOUT' COMMON BLOCK.
C
      RETURN
      END
```



5-AXLE TRACTOR-SEMI; 60 mph; Braking-in-a-Turn Example (+jackknife control)



5—AXLE TRACTOR—SEMI; 60 mph; Braking—in—a—Turn Example (+jackknife control)