DATA INTERCHANGE FORMAT FILES: A SIMPLE, DIRECT APPROACH TO PROVIDING TRANSPORTABLE GRAPHICS DATA

by

JAMES J. SHEEHY JR.

B.S., Ohio State University, 1975

A MASTER'S REPORT

submitted in partial fullfillment of the

requirements for the degree

MASTER OF SCIENCE

Department of Computer Science

KANSAS STATE UNIVERSITY
Manhattan, Kansas

1983

Approved by:

W. K. Hanley
Major Professor

CONTENTS

THIS BOOK CONTAINS NUMEROUS PAGES WITH THE ORIGINAL PRINTING BEING SKEWED DIFFERENTLY FROM THE TOP OF THE PAGE TO THE BOTTOM.

THIS IS AS RECEIVED FROM THE CUSTOMER.

THIS BOOK CONTAINS NUMEROUS PAGES WITH DIAGRAMS THAT ARE CROOKED COMPARED TO THE REST OF THE INFORMATION ON THE PAGE.

THIS IS AS RECEIVED FROM CUSTOMER.

LIST OF FIGURES

ACKNOWLEDGEMENTS

Recognition goes to Dr. William J. Hankley for his guidance and counsel and to Mr. Carlos Quales who gave freely of his time and knowledge on several different occasions. Recognition also goes to the remainder of the "LEAVENWORTH NINE", especially MAJ Donald D. Loftus, for willing and constant support and assistance. Finally, very special thanks and recognition goes to my wife Judy and my children Susan, James and Sean for their patience and understanding during this undertaking. Without their selfless, generous support this could not have been done.

Chapter I

INTRODUCTION

1.1 GENERAL

A recognized need for improved standards has long existed within the field of computer science. Of late, in the area of computer graphics, more and more emphasis has been placed upon developing standards which enhance data transportability in terms of machine, program and language independence. This report describes a method by which Data Interchange Format (DIF) files are used to improve data transportability and independence. The report details how DIF files are created on a host (Perkin-Elmer 8/32) computer and then transfered to an Apple II computer. Once transfered, the DIF file is used as a data input stream for the Visiplot/Visitrend graphics plotting program. This project has two purposes. The first is to examine DIF files in general. The second purpose is to demonstrate the manner in which data format standards enhance the exchange of data between applications programs which share a common language and to emphasise how data format standards facilitate both inter-machine and inter-language transfer of data. The DIF_BUILDER SYSTEM is utilized as a tool to demonstrate how DIF files accomplish these aims. The report is structured as follows:

1.) Chapter 1 provides background information on the

project along with an overview of the structure and content of the report. It highlights what was done, why it was done and then identifies central issues which were examined as the work on the project progressed. Finally, the chapter contains a synopsis of the findings resulting from the report and a list of definitions essential to the clear understanding of the chapter.

2.) Chapter 2 consists of a SYSTEM OVERVIEW which provides a general explanation of the DIF_BUILDER SYSTEM. The discussion begins with an overview of the system model and identifies each of the model's three main components. Chapter 2 concludes with a discussion of one of these main components: commercially available software. The discussion of commercial software packages examines applications programs which are compatible with data presented in the DIF format and also takes a brief look at a few of the more common terminal interface communications programs.

3.) Chapter 3 contains a in-depth discussion of DIF files. It examines how DIF files originated and what they offer in terms of utility. Chapter 3 also contains an explanation of the DIF file format and structure.

4.) Chapter 4 focuses on the implementation software which is unique to the DIF_BUILDER SYSTEM. Specifically it discusses the DIF_BUILDER program which creates the DIF files and the DIF.PURGE program which reformats DIF files after they have been transfered to the Apple II

computer.

5.) Chapter 5 contains suggestions for future improvements to the DIF_BUILDER SYSTEM and recommentations for possible future study in the area of DIF files.

This report also contains two appendices. Appendix A contains the program documentation for the DIF_BUILDER program and the DIF.PURGE program. Appendix B contains the DIF_TRANSFER User's Guide which will assist potential user's in implementing the system model.

1.2 CENTRAL ISSUES

During the design and implementation of the DIF_BUILDER SYSTEM, particular attention was focused on four central issues. These issues were consistent with the stated purpose of the project and were as follows:

1. FORMAT STRUCTURE: The structure of the DIF format was examined so as to assess its utility as a medium for enhancing data transportability.
2. STRENGTHS AND WEAKNESSES: the DIF format was evaluated so as to identify strengths and weaknesses peculiar to the use of DIF files.
3. INTER-MACHINE ENVIRONMENT: The impact of transfering DIF files between differing machines was examined.
4. COMMERCIAL SOFTWARE: An examination of commercially available software was conducted so as to identify some of the packages which are

capable of accepting data in DIF file format.

1.3 FINDINGS

The results of the project demonstrated the feasibility of transferring data between machines, programs and languages through the use of a standardized format. It further demonstrated that DIF files provide a viable standardized format and have the following advantages:

1. They fill a recognized need.
2. They are easily understood.
3. They are flexible.
4. They provide a standardized "end of data" feature.
5. They enhance program and language independency.

The project also exposed the following disadvantages associated with DIF files:

1. They are a "de-facto" standard.
2. They may require "after transfer" processing.
3. They may complicate editing.
4. They increase the opportunity for error.
5. They may make it more difficult to find errors.

Although the use of DIF files can increase the language and program independence of data, it does little to enhance the machine independence of data. This was evidenced by the need to reformat the DIF file after it had been transfered from one machine to another.

An examination of currently available software revealed

that there is a modest, but still growing, number of commercial software packages which can be used with DIF files. To continue this trend, Software Arts, Inc., the developers of the DIF format, have instituted the DIF Clearinghouse. The purpose of the clearinghouse is to provide a central agency for coordination of DIF related matters.

1.4 DEFINITIONS

1.) DATA INTERCHANGE FORMAT

A format developoed by Software Arts incorporated and designed to allow for the interchange of data between a variety of different programs.

2.) TRANSPORTABILITY

Transportability refers to the ease by which data can be transfered from one machine environment to another, from one program environment to another or from one language environment to another.

3.) MACHINE INDEPENDENCE

The degree to which the data is insensitive to the effects of various machine environments.

4.) PROGRAM INDEPENDENCE

The degree to which the data can be freely passed from one program to another. In this context the programs are considered to be written in the same language.

5.) LANGUAGE INDEPENDENCE

The degree to which the data can be freely passed between programs written in different languages.

6.) FIELD

The sub-elements of which a DIF file is constructed. A field may be from 1 to 256 bytes in length and is always terminated with a carriage return. In the DIF_BUILDER program fields have been arbitrarily limited to 25 bytes plus the carriage return.

CHAPTER II

THE SYSTEM MODEL

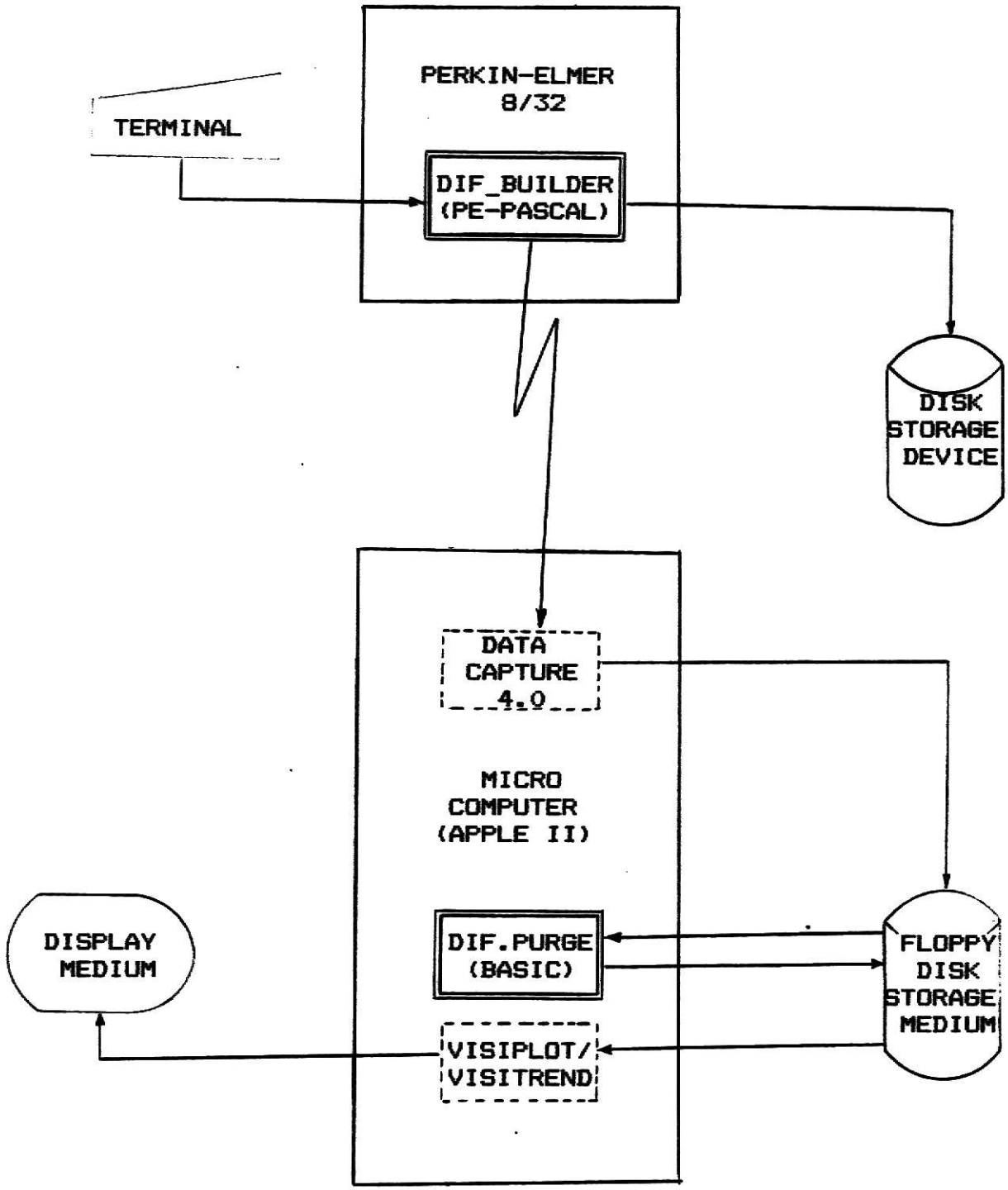
2.1 SYSTEM OVERVIEW

The the DIF_BUILDER SYSTEM enables a number of users to build unique data files while utilizing the facilities of a host computer which allows time-sharing. By this method several individual data files can be generated in parallel. The intent is to unburden the scarce resource, in this case the Apple II computer, which is then left free to display the data generated on the larger system. This approach affords a real advantage by reducing the total amount of time each user must spend using the scarce resource.

The DIF_BUILDER SYSTEM is designed around the following three elements:

1. Commercially available software
2. Applications programs which were specificaly written to support the DIF_BUILDER SYSTEM
3. The standardized DIF file format.

Figure 2.1 provides a graphic overview of the system. To accomplish its objectives the DIF_BUILDER SYSTEM employs a interactive program which is resident on the Perkin-Elmer 8/32 to create DIF files. The program uses a series of questions to lead the user through the construction of the DIF file. When the file is complete DIF_BUILDER automaticaly writes the newly created DIF file into the



LEGEND:

COMMERCIAL SOFTWARE -------

SYSTEM UNIQUE SOFTWARE ═══

DATA FLOW ————

FIG 2.1

user's disk file space. To transfer the file the user must access the Perkin-Elmer computer via the Apple II system. I elected to establish a data link between the two computers through the use of a commercially available software package. In this case the Data Capture Terminal Interface Program (TIP) was used. By implementing the Data Capture TIP on the Apple II, the user is able to transfer to the DIF file created on the host computer to an Apple II storage disk. The file is transfered as a text file. Once transfered the file must be reformated so that it is consistent with DIF files created on the Apple system. To do this the DIF.PURGE program was developed. Once reformated the the file is ready to be used as an input data stream for the Visiplot/Visitrend software package. The DIF format is the central feature of the system and serves to effect a bridge or link between the other two elements.

2.2 THE DIF FILE

Because of their relative importance DIF files will be discussed in detail in a separate chapter. See chapter 3.

2.3 SYSTEM-UNIQUE SOFTWARE

System-unique software will be discussed in detail in chapter 4. Appendix I.A contains the program listing for the DIF_BUILDER program. Appendix I.B contains the program listing for the DIF.PURGE program.

2.4 COMMERCIAL SOFTWARE

1. TERMINAL INTERFACE PROGRAM (TIP)

A TIP is a program which allows the user of a personal computer, such as the Apple II to interface with a larger host computer. The host is generally considered to be of the "mainframe" variety. The word "mainframe" is used here to connotate a computer sufficiently large enough to posses a sophisticated operating system and which normally provides service on a time-sharing basis. The TIP allows the user to access the resources of the host computer by establishing and maintaining a communications link with the host via a modem.

Currently there are several TIP's which are commercially available. Each of these programs provide the user with the ability to emulate other terminals which are "permanently" connected to the host. These programs vary in cost from $100.00 to $200.00 based upon their sophistication and capabilities. They offer features such as automatic dialing, software controlled modification of communications parameters and generation of ASCII characters which may not normally be available to the user. More importantly, these programs provide the user the ability to transfer, edit, transfer to the screen and write to storage text files passed from the host computer. Although these programs are generally equivalent, they may vary in areas such as the range of features offered, their degree of user friendliness and their data transfer rates. The following list contains the names of a few of the

currently available TIP's and their manufacturers: DATA CAPTURE, Southeastern Software Inc.; ASCII EXPRESS, Southwestern Data Systems; HAYES TERMINAL PROGRAM, Hayes Micrcomputer Products Inc. and TRANSCEND, SSM Microcomputer Products Inc.

The DATA CAPTURE program was used during this project. DATA CAPTURE was selected upon the basis of its availability and, although adequate, there are a number of other programs which would have served as well.

2. VISICALC .. VISIPLOT/VISITREND

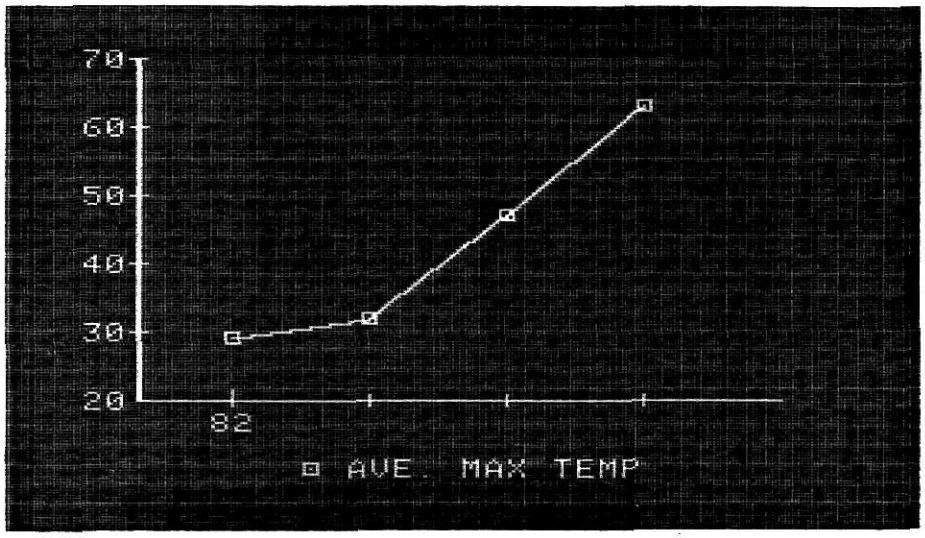
Visicalc and Visiplot/Visitrend are two of a series of commercial applications software programs which have been developed by Personal Software Inc. These packages are in the $150.00 to $300.00 price range and were developed for the small business and personal use computer markets.

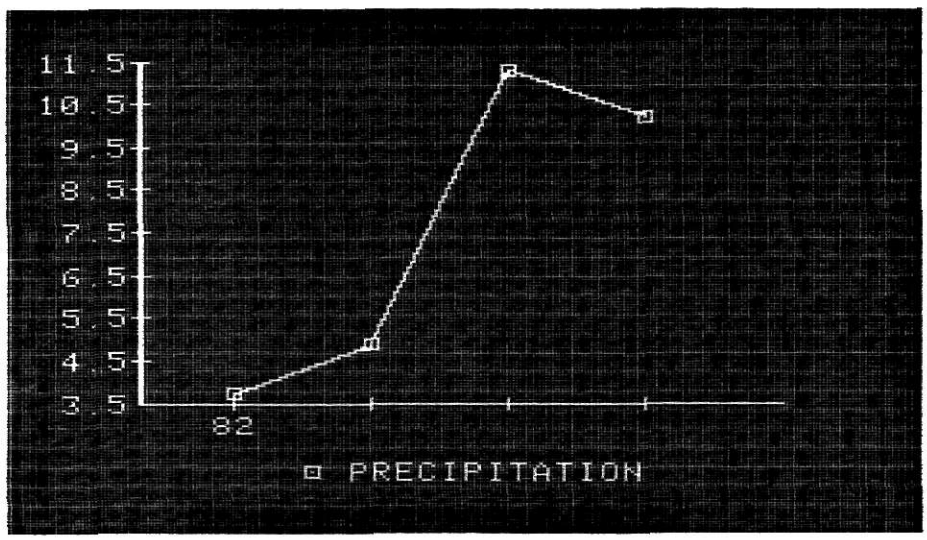
The Visicalc package offers the ability to enter data into a table or matrix type format. The matrix columns are ordered from "A" through "BK" and the rows are numbered from 1 through 254. The result is a matrix capable of holding 16002 individual data points. A matrix of this size serves to emphasise the need for a medium (such as a DIF filE) through which data can be entered quickly and efficiently. Visicalc also allows the user to enter equations which are unique to the user's need. Once entered along with the data, the user is able to edit, modify or expand the data as necessary. As a result the

final form of the data may be substantially different from its original form. When a session is complete the final form of the program matrix can be saved on a storage device for later use.

The Visiplot/Visitrend package consists of two distinct sub-packages. The Visiplot portion allows the user to display graphics images on a monitor or to reproduce them with on a printing device. The images that can be created are of a class normally associated with business graphics. As a result Visiplot has the capability to produce the following six types of charts: line, bar, area, hi-low, pie and scatter (see Figures 2.2 and 2.3). Each chart can be displayed individually or can be combined with other charts of the same type or of a different type; thus the user can create composite type charts. Visitrend is a statistical sub-package which develops data series for use with sophisticated analysis and forcasting techniques. Its features include moving average, data smoothing, percent of change, lead/lag and cumulative total functions.

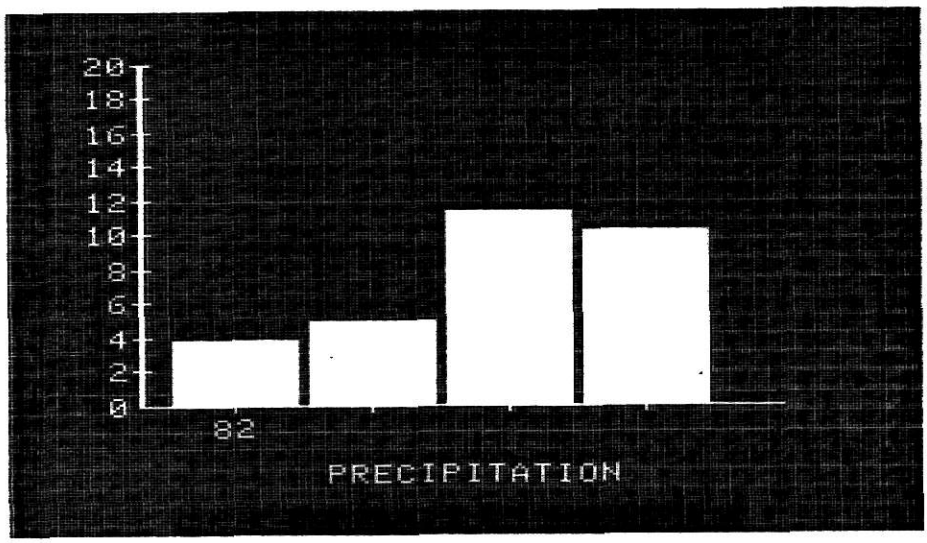
Both Visicalc and Visiplot/Visitrend are capable of reading and writing data which is in DIF format. This provides both of these packages with an extremely rich medium through which they can share data.





LINE CHARTS CREATED WITH THE VISIPLOT/VISITREND SOFTWARE PACKAGE

FIG 2.2



BAR AND AREA CHARTS CREATED WITH THE VISIPLOT/VISITREND SOFTWARE PACKAGE

FIG 2.3

CHAPTER III

DATA INTERCHANGE FORMAT FILES

3.1 GENERAL

DIF files are not purely a data base exchange format nor are they purely a graphics exchange format. Rather, the function of DIF files lies between these two extremes. As a result DIF files can best be defined a a data table exchange format. This type of format is particularily useful for providing data input streams to certain types of commercial software packages. Examples of commercial software for which DIF files have particular applicability include: "spread sheet" type packages such as Visicalc; business graphics plotting packages such as Visiplot and statistical analysis packages such as Visitrend.

The DIF file concept was developed by Software Arts incorporated in response to a widely recognized need for a standardized data exchange format. The need for an agreed-upon data exchange format becomes clear when the alternatives for transfering data in an environment lacking such a standard are examined. Currently there are three alternatives which may be employed, they are as follows:

1.) Create a new file in the correct format.

2.) Restucture the offending program to output the data in an acceptable format.

3.) Design separate programs to reformat offending files.

None of the alternatives offer an acceptable approach when

considered for use in a dynamic, multi-program environment. The first requires substantial duplication of effort; the second requires modifying otherwise suitable software. The final approach can lead to an excessive burden in terms of the development of programs to do the reformating and in terms of processing overhead. As an example, Figure 3.1 depicts an environment where there are five distinct programs, each of which requires a different data format. As a result ten separate data reformating programs are required if each program is to be able to use data files created for use by any of the remaining programs. To further emphasize this point the number of reformating programs required in an environment where there were 20 separate programs would increase by 19 times to a total of 190. This example serves to illustrate the growing need for an established, standardized data exchange format. DIF files serve to satisfy this need through the employment of an uncomplicated, flexible format. The primary appeal of DIF files is that they provide a data transfer medium that is flexible enough to have broad, general appeal and at the same time uncomplicated enough to be easily understood and implementated. Finally, it is by virtue of their uncomplex nature that DIF files are able to gain insensitivity to changes in their hardware and software environments.

3.2 HARDWARE ENVIRONMENT

The hardware environment can be separated into two

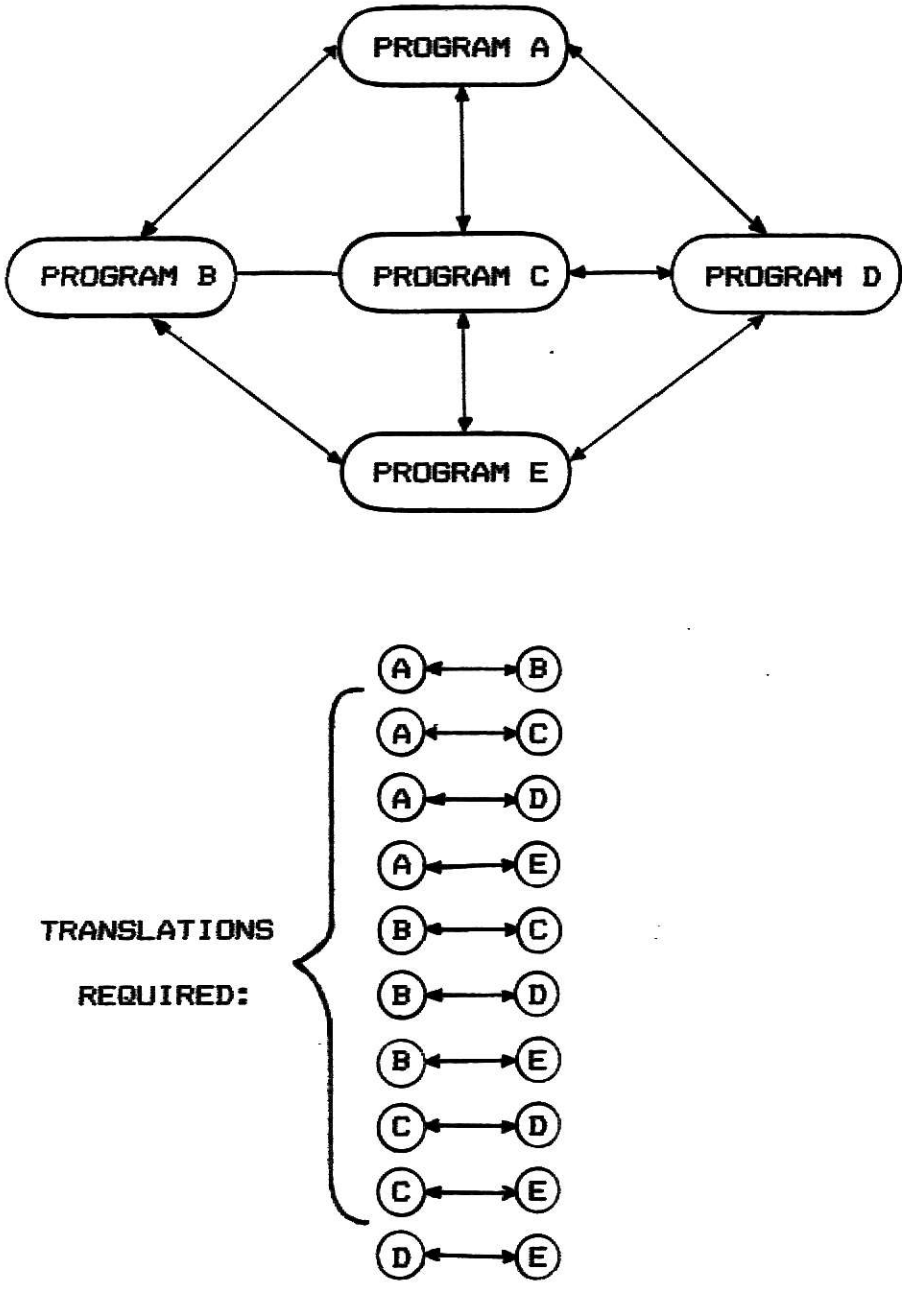

MULTI-PROGRAM
ENVIRONMENT
PROGRAM A
PROGRAM B
PROGRAM C
PROGRAM D
PROGRAM E
TRANSLATIONS
REQUIRED:
A B
A C
A D
A E
B C
B D
B E
C D
C E
D E
LEGEND:
PROGRAM A = UNIQUE PROGRAM
A B = DATA FORMAT TRANSLATION

FIG 3.1

distinct divisions. In the case where the machines are architecturally similar a homogeneous hardware environment is said to exist. However, if the machines are not architecturally equivilant then the hardware environment is said to be heterogeneous. In a heterogeneous environment a machine-level interface must be employed in order for the machines to communicate, regardless of the data transfer format employed. This would be the case if one machine utilized ASCII code while another machine used EBCDIC. Before these machines could be effectively linked an interface device would have to be implemented which could convert a bit stream from one type code to another. Again, this would be required irrespective of the data transfer format employed. In the case of homogeneous hardware, a translator at the machine-interface level is not required as long as the operating systems of both machines are identical. Where the operating systems are not idendical it may be necessary to process the input stream at some higher level to remove or modify inconsistent padding or control characters. From the preceding, it is apparent that DIF files do not enhance the efficiency of intercomputer data links. However, once the machines have been linked, the use of DIF files can significantly reduce or eliminate the need for further processing of the data prior to its use by a program. Further, if modifications are made to the machine environment DIF files will remain unaffected as long as the protocols at the machine-interface level are modified accordingly.

3.3 SOFTWARE ENVIRONMENT

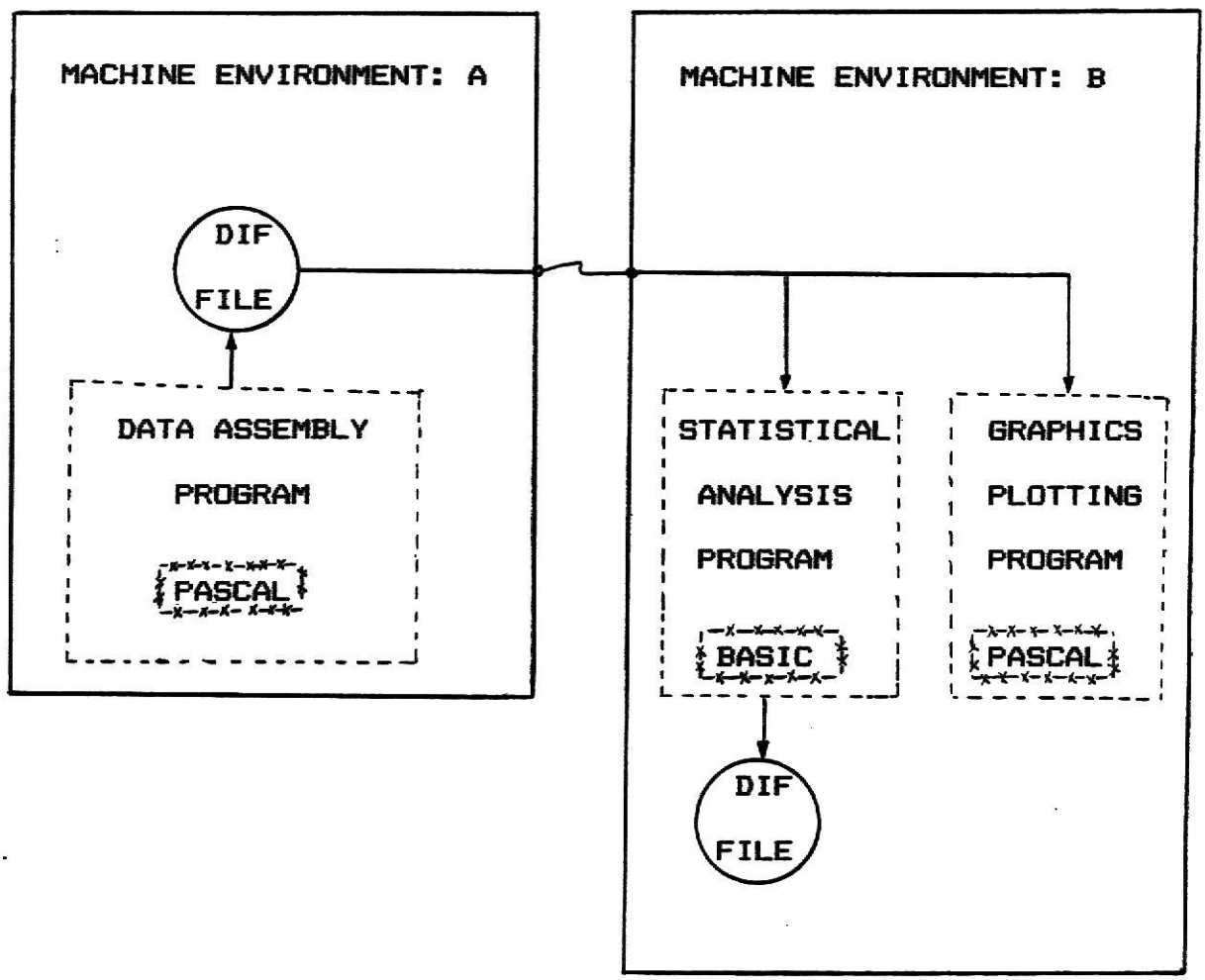
In the software environment DIF files exhibit a high degree of both program independence and language independence. From the standpoint of program independence DIF files effectively facilitate the transfer of data between programs designed for widely different purposes. As an example, the same DIF file could be used as a data source both for a graphics plotting program and a statistical analysis program (see Figure 3.2). In addition, if the output from the statistical analysis program was written into a DIF file, then that output could be passed to other programs which accepted DIF formated data. In the case of Figure 3.2, data from the statistical analysis package could be passed to the graphics display package. In addition to program independence, language independence is also gained through the use of DIF files. Once a program is structured to accept DIF files it will be able to process all properly formated files regardless of the parent language used to create the file. Figure 3.2 serves to illustrate this point by showing how a DIF file can be created in PASCAL on machine "A" and processed in BASIC or PASCAL on machine "B."

3.4 DIF FILE FORMAT

Much of the material in this section was extracted from [SOFT 80] and modified as necessary. For a more detailed treatment of this subject see [SOFT 80] and [CAND 81].

FILE TRANSFER

ENVIRONMENT



LEGEND:

——— = MACHINE ENVIRONMENT

- - - - = APPLICATIONS PROGRAM

-x-x-x- = PROGRAM LANGUAGE

○ = OUTPUT FILE

FIG 3.2

The DIF file format has the advantage of being both versitile and uncomplicated. Figures 3.3 through 3.8 will be used to assist in the explanation of the DIF file format. Figure 3.8 contains a diagramatic representation of the DIF file structure. Figure 3.3 represents a table containing data which is to be entered in a DIF file. Data in Figure 3.3 can be viewed in two different ways. When viewed from the perspective of a row, the data in the row is said to comprise a TUPLE. Each TUPLE is made up of all the data elements in a row and all rows must have the same number of data elements. A DIF file must have at least one TUPLE. When viewed from the perspective of a column the data in the column is said to comprise a VECTOR. All VECTORs must be of equal length. This means that that each VECTOR must possess the same number of data items as all other VECTORs in the file. The concept of VECTORs and TUPLEs is centeral to the construction of DIF files. DIF files are divided into two major parts, the HEADER PART and the DATA PART (see Figure 3.4).

3.5 THE HEADER PART

The HEADER PART is composed of a number of three field elements called HEADER ITEMS. The purpose of a HEADER ITEM is to relay special information about a particular DIF file to the using program. The three field structure used by each HEADER ITEM is shown in Figure 3.5.

RAW DATA

TABLE

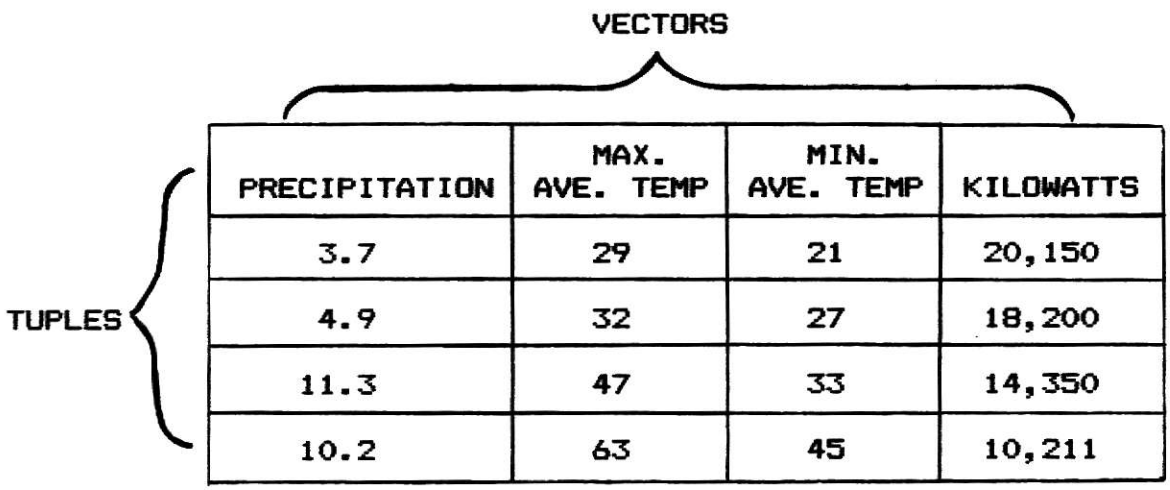


FIG 3.3

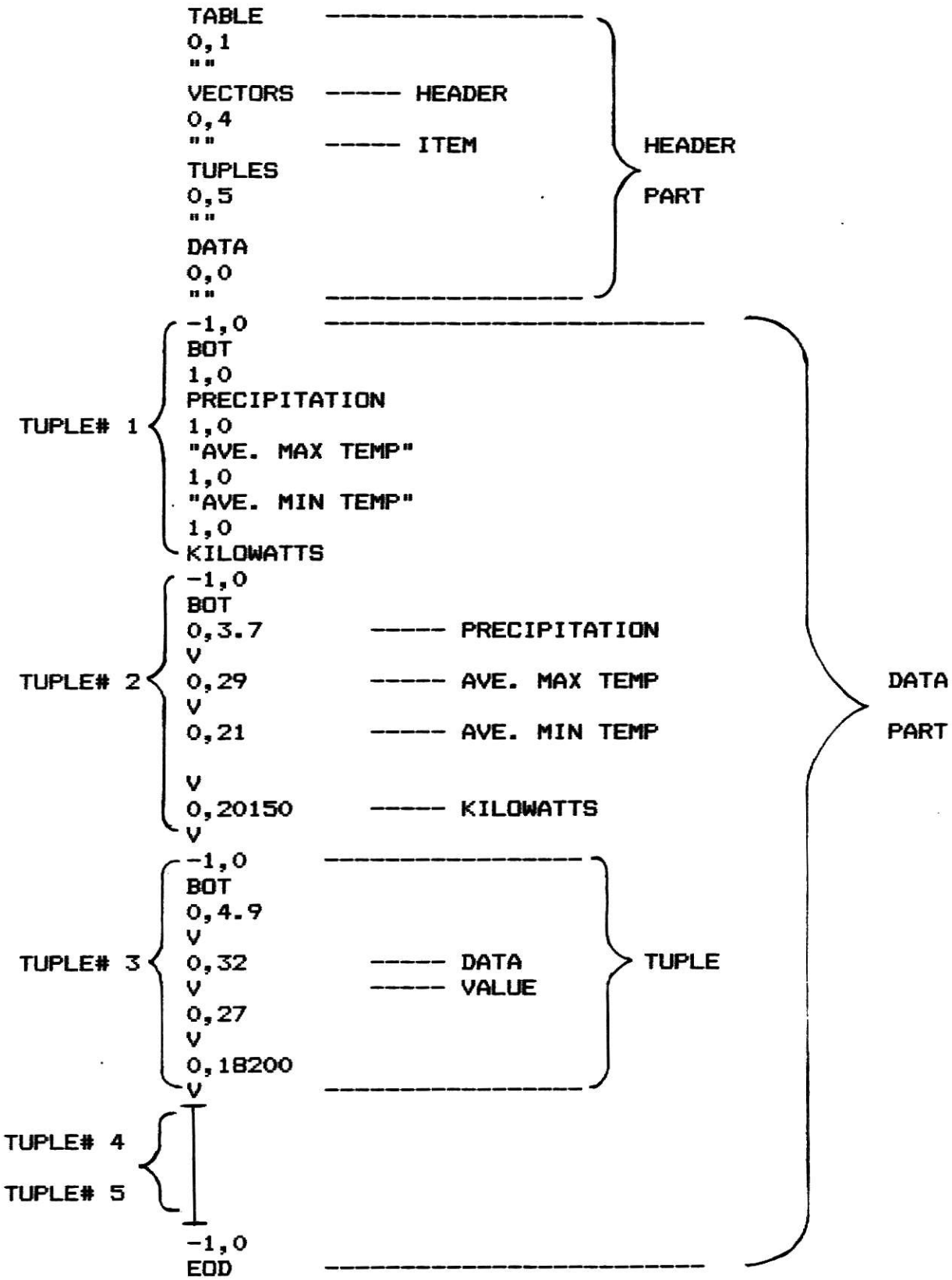


TABLE
0,1
""
VECTORS
0,4
""
TUPLES
0,5
""
DATA
0,0
""
----- HEADER
----- ITEM
HEADER
PART
-1,0
BOT
1,0
PRECIPITATION
1,0
"AVE. MAX TEMP"
1,0
"AVE. MIN TEMP"
1,0
KILOWATTS
TUPLE# 1
-1,0
BOT
0,3.7
V
0,29
V
0,21
V
0,20150
V
TUPLE# 2
----- PRECIPITATION
----- AVE. MAX TEMP
----- AVE. MIN TEMP
----- KILOWATTS
-1,0
BOT
0,4.9
V
0,32
V
0,27
V
0,18200
V
TUPLE# 3
----- DATA
----- VALUE
TUPLE
TUPLE# 4
TUPLE# 5
-1,0
EOD
DATA
PART

FIG 3.4

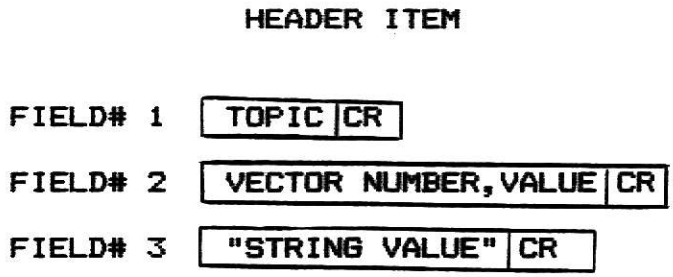


FIG 3.5

The TOPIC is a keyword which occupies The first field in each HEADER ITEM and which identifies the purpose of the HEADER ITEM. The VECTOR NUMBER is used to identify the specific VECTOR to which a particular HEADER ITEM applies. If the HEADER ITEM applies to more than one VECTOR then a "0" is entered in the VECTOR NUMBER location. The VALUE is an integer which conveys numerical information concerning the HEADER ITEM. As an example, the VALUE associated with a HEADER ITEM which has "TUPLES" as a TOPIC identifier, would be used to convey how many TUPLEs are in the file. The STRING VALUE occupies its own unique field and is used to associate input string data with a particular TOPIC. In sum, each HEADER ITEM consists of three fields, each of which is terminated by a carriage return. The first field in each HEADER ITEM contains the TOPIC. The second field contains the VECTOR NUMBER and the VALUE separated by a comma. If the VECTOR NUMBER or VALUE are not used, a "0" is entered in the appropiate location. When the STRING VALUE is not used double

quote marks are entered in the string field. Two TOPICS, TABLE and DATA, are required in all DIF files. The remaining items are optional. A brief description of the most common header items follows:

1. TABLE ..... This is the first entry in the file. It identifies the file as a table of data. This is a required HEADER ITEM in all programs.
2. VECTORS ... Tells how many VECTORs are in the data file. Some programs may treat VECTORS as a required TOPIC. In addition, this item must appear in the file prior to any HEADER ITEMS which reference VECTORs by number.
3. TUPLES .... This item gives the count of TUPLEs in the file.
4. LABEL ..... Provides the user an optional method with which to label the VECTORs.
5. DATA ..... Alerts the program that the HEADER PART is ending and the DATA PART is begining.

3.6 THE DATA PART

The second major division of the file consists of the DATA PART. The DATA PART is made up of TUPLEs, each of which has its own set of VECTOR values. As illustrated in Figure 3.4 the first TUPLE consists of data taken from the first row in Figure 3.3. The data in the second row is in TUPLE# 2, etc. TUPLEs are made up of one or more two-field

groupings; these groupings are called DATA VALUEs. The first of these fields serves a dual purpose. It indicates the type of data to be found in the DATA VALUE and it is used to pass numeric data. The second field in the DATA VALUE is used to pass string data. Figure 3.6 is a diagram of the DATA VALUE structure.

DATA VALUE

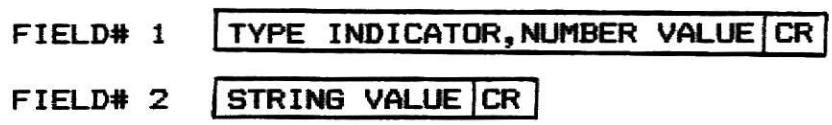


FIG 3.6

FIELD# 1 has two elements, the first of these is the TYPE INDICATOR element. The TYPE INDICATOR requires an integer value and act as a four position flag. The flag indicates to the program how to interpret the NUMBER VALUE and the STRING VALUE associated with each DATA VALUE. The four valid TYPE INDICATOR entries are as follows:

1. (-1)... Indicates the DATA VALUE has a special status. Special status DATA VALUEs are used to mark either the begining of a TUPLE or the end of the data file.
2. (0).... Indicates that the data stored in the DATA VALUE is numeric and will be found in

the NUMBER VALUE location.

3. (1).... Indicates that the data stored in the DATA VALUE is in the form of a string and can be found in the STRING VALUE field.
4. (2).... Indicates the data stored in the DATA VALUE is application specific and may have some special meaning to programs which share the data.

The second element in FIELD# 1 is the NUMBER VALUE. A numeric value is always entered in this location. The number may be preceded by a unary operator (+,-), it may have a decimal and may be followed by the letter "E" and a base ten exponent. The exponent may also be preceded by a unary operator.

FIELD# 2 is used for string values. If the TYPE INDICATOR is set to "0", indicating the data is numeric, then the letter "V" is placed in the string value field. Under these conditions the letter "V" is not enclosed in quotes. The DIF format also provides the user the ability to override the value that is located in the NUMBER VALUE field. This is done by placing a VALUE INDICATOR in the STRING VALUE field even though the TYPE INDICATOR is set to "0", a condition which indictes that the DATA VALUE contains numeric data (see Figure 3.7).

DATA VALUE

FIELD# 1

TYPE INDICATOR	NUMBER VALUE	CR
0	, 4525	cr

FIELD# 2

STRING VALUE	CR
NA	cr

FIG 3.7

In Figure 3.7 "NA" is a VALUE INDICATOR which instructs the program to ignore the NUMBER VALUE stored in FIELD# 1. VALUE INDICATORs can be viewed as embeded instructions which provide the user the flexibility to influence how a program treats selected NUMBER VALUEs. VALUE INDICATORs can be user defined but should be registered with the DIF Clearinghouse. VALUE INDICATORs are not enclosed in quotes.

If the TYPE INDICATOR is set to "1", indicating that the DATA VALUE contains string data, then the input will be located in the STRING VALUE field. Quote marks are only required when input strings contain blank spaces. TUPLE# 1 in Figure 3.4 illustrates the use of STRING VALUEs.

All TUPLEs are made up entirely of a set of DATA VALUEs. In this regard, there are two special cases of DATA VALUEs which require examination. These are the

DATA VALUEs which mark either the begining of a TUPLE or the end of the input data. Special case DATA VALUEs can be easily identified because they always have a TYPE INDICATOR entry of "-1" and a NUMBER VALUE of "0." When the intent is to mark the begining of a TUPLE, then "BOT" is placed in the DATA VALUEs STRING VALUE location. When used to mark the end of data then "EOD" is placed in the STRING VALUE location.

3.7 CONCLUSIONS

As indicted by their structure and format, DIF files provide a powerful medium for the transfer of data. A list of some of the advantages gained from the use of DIF files includes:

1. DIF files fill a recognized need for a data exchange format.
2. The DIF file format is both uncomplicated and easy to understand.
3. DIF files are flexible enough to be adapted to specalized situations.
4. The "EOD" feature provides a graceful way in which to mark the end of the data file.
5. DIF files enhance language and program independence.

Disadvantages of DIF files include:

1. DIF files are only a "de-facto" standard.
2. Some processing of DIF files may be required due to differences in operating systems.

3. If necessary, editing of DIF files may require more effort than when the DIF format is not used.

4. Because a DIF file may "fail to load" as a result of either inconsistancies in data or inconsistancies in file structure, there is a greater chance for errors. As a result, errors that do occur may be more difficult to locate.

On balance, DIF files provide an exceptional tool for the transfer of data. They represent a large "first step" in the direction of establishing a much needed industry standard. Currently, the number of software developers who are using the DIF format is steadily increasing. This indicates that the economic as well as operational advantages of DIF files are begining to be recognized throughout the industry. While DIF files may never become an accepted, industry-wide data exchange standard, they hold the potential for providing a solid basis for such a standard should one be developed. To this end, Software Arts, Inc. has established the DIF Clearinghouse for the purpose of coordinating information concerning DIF files. For further information concerning DIF files, correspondence can be sent to:

DIF Clearinghouse
P.O. Box 527
Cambridge, MA 62139

```
1.  <DIF FILE>          ::= <HEADER PART>:<DATA PART>

2.  <HEADER PART>       ::= 2{<HEADER ITEM>}

3.  <HEADER ITEM>       ::= <H-FIELD#1>:<H-FIELD#2>:<H-FIELD#3>

4.  <H-FIELD#1>         ::= <TOPIC>:<CR>

5.  <H-FIELD#2>         ::= <TYPE INDICATOR>:<COMMA>:<NUMBER
                            VALUE>:<CR>

6.  <H-FIELD#3>         ::= <QUOTE>:<STRING>|<NULL>:<QUOTE>:<CR>

7.  <DATA PART>         ::= 1{<TUPLE>}

8.  <TUPLE>             ::= <DATA VALUE>:1{VECTOR}:<DATA VALUE>

9.  <VECTOR>            ::= 1{<DATA VALUE>}

10. <DATA VALUE>        ::= <D-FIELD#1>:<D-FIELD#2>

11. <D-FIELD#1>         ::= *<MINUS>:<TYPE INDICATOR>:<COMMA>:
                            <NUMBER VALUE>:<CR>

12. <D-FIELD#2>         ::= *<QUOTE>:<STRING>|<VALUE INDICATOR>:
                            *<QUOTE>:<CR>

13. <TOPIC>             ::= {<ALPHAS>}

14. <NUMBER VALUE>      ::= *<OPERATORS>:1{<DIGITS>}:*[<EXPONENTS>:
                            *<OPERATORS>:1{<DIGITS>}]

15. <STRING>            ::= ^{^<ALPHA>||^<DIGIT>||^<CHAR>}

16. <DIGIT>             ::= 0|1|2|....|9

17. <ALPHA>             ::= A|B|C|....|Z

18. <CHAR>              ::= #|$|%|....|?

19. <COMMA>             ::= ,

20. <TYPE INDICATOR>    ::= 0|1|2
```

FIG 3.8

(CONTINUES ON NEXT PAGE)

21. <OPERATOR> ::= <MINUS>| +

22. <MINUS> ::= -

23. <EXPONENT> ::= E

24. <CR> ::= CARRIAGE RETURN

25. <NULL> ::= EMPTY SET

LEGEND:

|| = AND

| = OR

* = 0 OR 1

{} = n OR MORE (WHERE n IS SOME NUMBER)

^ = 0 OR MORE

<> = NON-TERMINAL

: = IS FOLLOWED BY

FIG 3.8 (CONT.)

CHAPTER IV

SYSTEM UNIQUE SOFTWARE

4.1 GENERAL

In addition to the commercial software packages, it was necessary to develop two applications programs which were unique to the DIF BUILDER SYSTEM. The first of these programs, refered to as DIF_BUILDER, is designed to construct DIF files from data supplied by the user. The DIF_BUILDER program resides on the Perkin-Elmer 8/32. The second program developed for the DIF_BUILDER SYSTEM, DIF.PURGE, resides on the Apple II computer and is used to reformat the DIF file (see Figure 4.1). For listings of these programs see Appendix A.

4.2 DIF_BUILDER PROGRAM

The DIF_BUILDER program resides on the host computer, its function is to construct DIF files. The program employes Perkin-Elmer PASCAL as its base language and is implemented on a Perkin-Elmer 8/32 computer which operates under the OS/32 operating system. DIF_BUILDER is a interacitive program which can be run by invoking a CSS file. The CSS filename is BUILDER. When run, this program creates a text file. The text file is constructed in the DIF file format and then written into the user's filespace. As one of its features, DIF_BUILDER automatically name the

THE DIF_BUILDER
ENVIRONMENT

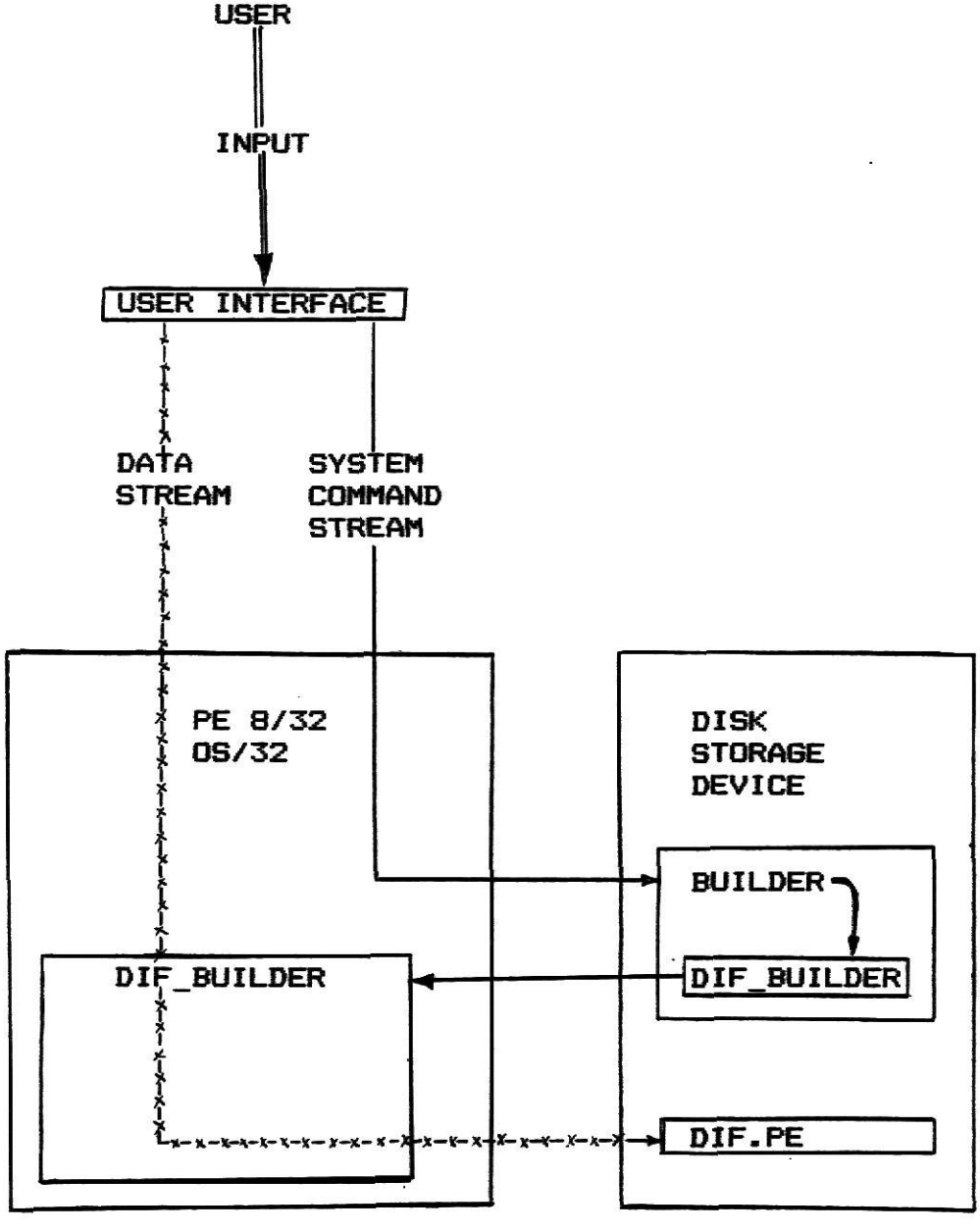


FIG 4.1

file it creates. In all cases the new file will be identified as DIF.PE (DATA INTERCHANCE FORMAT.PERKIN-ELMER). The user's can access, edit or transfer a given DIF file by refering to DIF.PE. In addition, the option to rename the file anytime after its construction is available through the use of the rename command.

4.3 DIF_BUILDER STRUCTURE

DIF_BUILDER is comprised of six distinct procedures. These procedures are INIT_FILE, HEAD_BUILDER, DATA_BUILDER, CONVERTER, READ_TERM and WRITE_FILE (see Figure 4.2). A brief description of the functions of each procedure follows.

1. INIT_FILE: The purpose of INIT_FILE is to establish the program's interactive relationship with the user. It accomplishes this by providing the user with a brief introduction to the program and its purpose. The user is then given the opportunity to continue when he is ready or to terminate if he so desires. If the user elects to continue, INIT_FILE opens a file into which DIF formated output can be written.
2. HEAD_BUILDER: This procedure is called from INIT_FILE. The purpose of HEAD_BUILDER is to construct the DIF file HEADER PART. DIF_BUILDER employs only the four, most common, TOPIC IDENTIFIERS (TABLE, VECTORS, TUPLES, DATA).

PROGRAM STRUCTURE

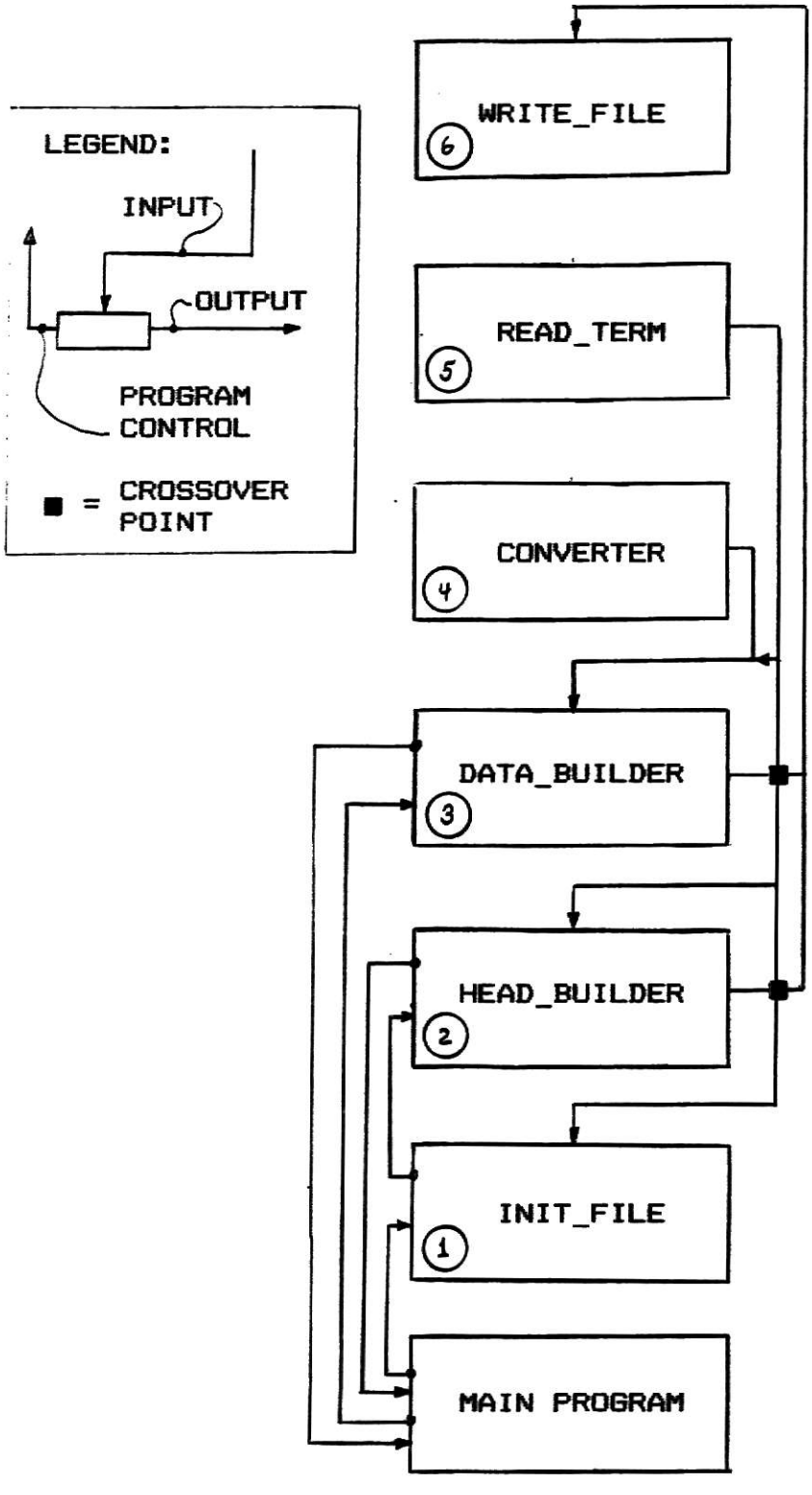


FIG 4.2

Although these are more than adequate in most cases, this restriction may, under certain conditions, limit the ability of the user to tailor a given file to his particular needs. If desired the program could be restructured to allow for the addition of other TOPIC IDENTIFIERS to the HEADER PART. In addition to allowing for the introduction of the two required TOPIC IDENTIFIERS into the file, DIF_BUILDER also allows the user to enter the number of VECTORs and TUPLEs from which the DIF file will be constructed. This not only provides a control mechanism for the construction of the file but also enables certain using programs to calculate the amount of memory storage required to process the data file. In all cases the number of VECTORs and the number of TUPLEs is limited to between 1 and 99. Minor modification to this procedure and the CONVERTER procedure would be required to alter these limits. File output from HEAD_BUILDER is passed to the WRITE_FILE procedure for inclusion in the DIF file under construction. Upon completion of this procedure control reverts to the main program.

3. DATA_BUILDER: This procedure is called from the main program; its purpose is to construct the

DATA PART of the DIF file. The procedure begins by automatically constructing a field with a SPECIAL DATA VALUE, this is followed by a second field which contains a STRING VALUE of "BOT." This is a special case of the DATA VALUE which marks the begining of all TUPLEs. Following this the procedure interactively queries the user to enter the (data) TYPE INDICATOR and the (data) NUMBER VALUE or the (data) STRING VALUE for each VECTOR in the TUPLE. Requests for input data are identified to the user in TUPLE, VECTOR order. DATA_BUILDER constructs fields which are 25 charactes long; any excess field spaces are padded with blanks. As each field is completed it is sent to WRITE_FILE for inclusion in the file under construction. Finally, DATA_BUILDER automatically inserts the "EOD" DATA VALUE when it determines that data has been recieved for all TUPLEs and VECTORs. When completed, control returns to the main program and the program terminates.

4. CONVERTER: This procedure is called by HEAD_BUILDER. The purpose of CONVERTER is to convert given input from character to integer values. Once the conversion is complete, program control returns to HEAD_BUILDER where the integer numbers are used to monitor the

count of VECTORs and TUPLEs created by the program.

5. READ_TERM: Procedure READ_TERM is accessed by several procedures within the program. It is used to "read in" an 80 character input line from the terminal and then pass the line to the calling procedure.

6. WRITE_FILE: This procedure is accessed by two other procedures, HEAD_BUILDER and DATA_BUILDER. It serves two primary functions. First it edits each input line by removing all unnecessary blanks and then adding a carriage return at the end of the useable data in each field. This reduces the total number of bytes which are contained in the file and serves to create a field which is identical to the concept of the field format as discused in CHAPTER III. Second, WRITE_FILE writes the field, which is now in its final form, into DIF.PE. Finally, WRITE_FILE can be used by any new procedures which might be added to DATA_BUILDER. Thus, procedures could collect data other than directly from the user. These procedures could be designed so as to write their data into arrays of type character. After the construction of each array it would be passed to WRITE_FILE for inclusion in the file under

construction. Figure 4.3 provides an example of DIF.PE as it would appear after being written into the user file space.

4.4 DIF.PURGE PROGRAM

The DIF.PURGE program reformats DIF files which have been transfered from the host computer to the Apple II computer. This reformating is required because the OS/32 operating system adds special padding characters to each DIF file field as the field is being created (see Figure 4.3). The unwanted characters consist of a series of three underscore characters, a dash and then two more underscore characters in front of each field. An additional underscore is placed at the end of each field just prior to the carriage return. These padding characters are suppresed by the OS/32 operating system and thus are never seen by the user. However, since they cannot be ignored by the Apple II nor by any of the software which uses DIF files, these characters must be removed. DIF.PURGE accomplishes this by individually examining each character to determine if it is a padding character and then retaining or discarding the character as necessary.

4.5 DIF.PURGE LOGIC

The DIF.PURGE program is interactiv only to the extent that it requires the user to specify the input filename (DIF.PE unless previously changed by the user) and output

```
__-__TABLE_
___-__0,1_
___-__""_
___-__VECTORS_
___-__0,4_
___-__""_
___-__TUPLES_
___-__0,5_
___-__""_
___-__DATA_
___-__0,0_
___-__""_
___-__-1,0_
___-__BOT_
___-__1,0_
___-__PRECIPITATION_
___-__1,0_
___-__"AVE. MAX TEMP"
___-__1,0_
___-__"AVE. MIN TEMP"
___-__1,0_
___-__KILOWATTS_
___-__-1,0_
___-__BOT_
___-__0,3.7_
___-__V_
___-__0,29_
___-__V_
___-__0,21_
___-__V_
___-__0,20150_
___-__V_
___-__-1,0_
___-__BOT_
___-__0,4.9_
___-__V_
___-__0,32_
___-__V_
___-__0,27_
___-__V_
___-__0,18200_
___-__V_
___-__-1,0_
___-__BOT_
___-__0,11.3_
___-__V_
___-__0,47_
___-__V_
___-__0,33_
___-__V_
___-__0,14350_
___-__V_
```

FIG 4.3
(CONTINUES ON NEXT PAGE)

```
___-__-1,0_
___-__BOT_
___-__0,10.2_
___-__V_
___-__0,63_
___-__V_
___-__0,45_
___-__V_
___-__0,10211_
___-__V_
___-__-1,0_
___-__EOD_
___-___
```

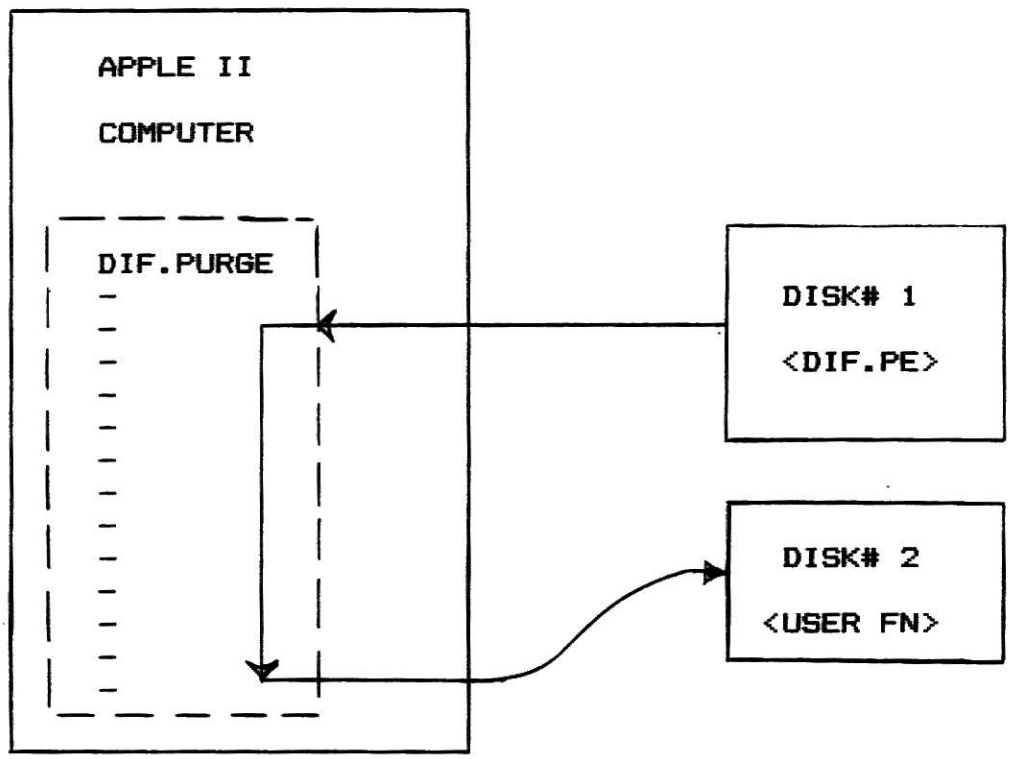
FIG 4.3 (CONT.)

filename. The user can rename his file at this point. DIF.PURGE has been designed to operate on a system which is supported by at least two disk drives. As a result it expects to read the input file from disk drive# 1 and write an output file to disk drive# 2 (see Figure 4.4).

DIF.PURGE performs a character by character examination of the input file. In every instance it tests to see to see if the character to be removed is either a dash or an underscore. The program assumes all underscores are unwanted, thus it discards all such characters. However, because a dash (or minus) frequently assumes a valid role in the construction of a DIF file, DIF.PURGE must perform a seperate test each time a dash is located. This is done by keeping track of whether or not a dash has already been removed from the field that is currently being scanned. Since there is only one offending dash in each field and this is always the first dash encountered, it is possible to determine the validity of a dash contingent upon whether or not a dash has been previously removed from the field being scanned. If a dash has already been removed from the field the "suspect dash" is retained. If a dash has not been removed from the current field then the "suspect dash" is purged.

When it is determined that a character is valid that chacter is stored in memory. After each character in the file has been examined DIF.PURGE opens a new file and then writes the valid character stream into the new file. Upon

THE DIF.PURGE ENVIRONMENT



LEGEND: <FILENAME>

FIG 4.4

completion of the write process the program terminates. Figure 4.5 depicts a DIF file after the file has been processed by DIF.PURGE.

4.6 ASSESSMENT OF UNIQUE SOFTWARE

Both programs were instrumental in contributing to the accomplishemnt of the stated intent of the project. First, DIF_BUILDER made it possible to create a DIF file. Next, DIF.PURGE enabled the file to be run in a machine environment different from the one in which it was created. Both of these functions were supportive of the overall objective of creating and transfering DIF files. In addition, the development of these programs contributed either directly or indirectly to the assessment of the specified central issues:

1. Assess the utility of DIF files for use as a data transfer medium.
2. Examine software which uses DIF files.
3. Ascertain the strengths and weaknesses of DIF files.
4. Assess the impact of operations involving DIF files when conducted in an inter-machine environment.

Although both programs exceeded the minimum requirements necessary to accomplish their stated purpose, they could both be enhanced through the implementation of various modifications. DIF_BUILDER, could be improved by

```
TABLE
0,1
""
VECTORS
0,4
""
TUPLES
0,5
""
DATA
0,0
""
-1,0
BOT
1,0
PRECIPITATION
1,0
"AVE. MAX TEMP"
1,0
"AVE. MIN TEMP"
1,0
KILOWATTS
-1,0
BOT
0,3.7
V
0,29
V
0,21
V
0,20150
V
-1,0
BOT
0,4.9
V
0,32
V
0,27
V
0,18200
V
-1,0
BOT
0,11.3
V
0,47
V
0,33
V
0,14350
V
```

FIG 4.5
(CONTINUES ON NEXT PAGE)

```
-1,0
BOT
0,10.2
V
0,63
V
0,45
V
0,10211
V
-1,0
EOD
```

FIG 4.5 (CONT.)

enriching the number of TOPIC IDENTIFIERS which it supports. Further, the program lacks the sophisticated error handling routines necessary if a program is to be considered "user friendly." DIF.PURGE could be converted to a structured language such as PASCAL. In addition, DIF.PURGE should be integrated with the terminal interface program. This should be done to reduce the number of steps in the file transfer process and to make the operation of DIF.PURGE transparent to the user. Finally, because there are rare occasions when an underscore may appear within a STRING VALUE of a given DIF file, DIF.PURGE should be modified to enable it to test each underscore for validity.

CHAPTER V

RECOMMENDATIONS

5.1 PROGRAM EXPANSION

The DIF_BUILDER SYSTEM was designed as an instrument through which DIF files could be studied. As a result, not all features of the DIF format were incorporated into the file construction program. This shortcoming could be corrected by expanding DIF_BUILDER to include all DIF file features as discussed in [SOFT 80].

5.2 FUTURE STUDY

Although DIF files provide a powerful method by which to transfer common data streams (such as a stream of numbers to be processed by a statistical analysis program), it is possible that their scope of usefulness may extend other more complex areas. In particular, DIF files may prove to be a worthy medium for transporting data streams made up of program elements or primitive commands. Two unique possibilities come to mind. First, in the area of computer graphics, DIF files could be used to transfer series of primitive commands which could be used by a recieving program to recreate an image created on another machine. The implication is that a larger machine would perform the heavy processing associated with creating the original image and writing the primitives into a DIF file. A smaller machine would recieve and display the image.

A second area for consideration would be to study the feasibility of using the DIF format to transport programs from machine to machine. The receiving program would convert the text input from the DIF file into a tokanized output stream which could be written into a separate file. After completion the tokanized file could be run as a program.

BIBLIOGRAPHY

[SOFT 80] Software Arts Inc., Programmer's Guide to DIF, Software Arts Technical Note: SATN-18, 1980

[CAND 81] Calish, C.E., and Mayer, M.F., DIF: A Format for Data Exchange Between Applications Programs, Byte, vol. 6, no. 11, pp. 174-206, November 1981

[AHL 81] Ahl, D.H., Evaluation of Visitrend and Visiplot from Personal Software, Creative Computing, vol. 7, no. 12, pp. 98-104, December 1981

[VISI 81] User's Guide for Apple II & II Plus, 48K 16 Sector, Software Arts Inc., 1981

REFERENCES

1. Lelbson, S., The Input/Output Primer Part 1: What is I/O?, Byte, vol. 7, no. 2, February 1982.

2. Lelbson, S., The Input/Output Primer Part 2: Interrupts and Direct Memory Access, Byte, vol. 7, no. 3, March 1982.

3. HP, With Software Packages, Lets Desktop Line Communicate With Firm's Mini's, Data Communications, vol. 10, no. 6, June 1981.

4. Private, Public Link is Videotex Key, Data Communications, vol. 10, no. 5, May 1981.

5. Reintjes, P.B., Network Tools, Ideas for Intelligent Network Software, Byte, vol. 6, no. 10, October 1981.

6. Loingfield, M., Build an Intercomputer Data Link, Byte, vol. 6, no. 4, April 1981.

7. Saal, H.J., Local Area Networks, Byte, vol. 7, no. 10, October 1981.

8. Malone, J., The Microcomputer Connection to Local Networks, Data Communications, vol. 10, no. 12, December 1981.

9. Klein, M., Files on Parade Part 1: Type of Files, Byte, vol. 4, no. 2, February 1979

10. Klein, M., Files on Parade Part 2: Using Files, Byte, vol. 4, no. 3, March 1979.

11. Paring a Local Net with Some Apples, Data Communications, vol. 10, no. 1, January 1981.

12. Communications and Networking Invade the Home Front, Data Communications, vol. 10, no. 2, February 1982.

13. Data Capture 4.0 Documentation, Southeastern Software, 1980

14. The DOS Manual, Apple Computer Inc., 1980.

15. Apple II Reference Manual, Apple Computer Inc., 1979.

16. Applesoft Tutorial, Apple Computer Inc., 1979.

17. BASIC Programming Reference Manual, Apple Computer Inc., 1978.

18. Perkin-Elmer PASCAL User's Guide, Language Reference and Run Time Support Reference Manual, Edward Sherman Brown Data Systems Group, 1979.

APPENDIX A

A1

DIF_BUILDER PROGRAM LISTING

```
  PROGRAM DIF_BUILDER ( OUTPUT , TERM_IN , DIF_OUT )

(* DIF_BUILDER IS AN INTERACTIVE PROGRAM   *)
(* WHICH BUILDS DIF FILES FROM USER INPUT  *)

  ; CONST ITEM_LIMIT = 4
    ; HEAD_1 = 'TABLE                    '
    ; HEAD_2 = 'VECTORS                  '
    ; HEAD_3 = 'TUPLES                   '
    ; HEAD_4 = 'DATA                     '
    ; HEAD_5 = '0,1                      '
    ; HEAD_6 = '0,0                      '
    ; HEAD_Q = '""                       '
    ; DATA_1 = '-1,0                     '
    ; DATA_2 = 'BOT                      '
    ; DATA_3 = '0,                       '
    ; DATA_4 = 'V                        '
    ; DATA_5 = '1,0                      '
    ; DATA_6 = 'EOD                      '
    ; MAXLN = 80
    ; STUB_SIZE = 2
    ; MAX_ENTRY_LEN = 25
    ; TEN = 10

  ; TYPE ENTRY_INDEX = 1 .. MAX_ENTRY_LEN
    ; TERM_INDEX = 1 .. MAXLN
    ; STUB_INDEX = 1 .. STUB_SIZE
    ; TERM_LN = ARRAY [ TERM_INDEX ] OF CHAR
    ; ENTRY_LN = ARRAY [ ENTRY_INDEX ] OF CHAR
    ; STUB = ARRAY [ STUB_INDEX ] OF CHAR
    ; DIF_FILE = FILE OF  ENTRY_LN

  ; VAR INT_NUM : INTEGER
    ; TERM_IN : TEXT
    ; TERM_OUT : TERM_LN
    ; DIF_OUT : TEXT
    ; VECTORS , TUPLES : CHAR
    ; TEMP_VEC , TEMP_TUP : STUB
    ; DONE_FLAG : BOOLEAN
```

```
  ; PROCEDURE WRITE_FILE ( DIF_LINE : ENTRY_LN )

(* THE WRITE_FILE PROCEDURE HAS SOLE RE-    *)
(* SPONSIBILITY FOR WRITING INTO THE DIF    *)
(* FILE SPACE                               *)

    ; VAR DIF_INDEX , OUT_INDEX : ENTRY_INDEX
      ; STOP_INDEX : STUB_INDEX
      ; STOP_TST : STUB
      ; OUT_DONE : BOOLEAN

    ; BEGIN
        OUT_DONE := FALSE
      ; OUT_INDEX := 1
      ; STOP_INDEX := 1
      ; DIF_INDEX := 1
      ; REPEAT
        ; DIF_INDEX := OUT_INDEX
        ; FOR STOP_INDEX := 1 TO STUB_SIZE
          DO BEGIN
            STOP_TST [ STOP_INDEX ] := DIF_LINE [ DIF_INDEX ]
          ; DIF_INDEX := SUCC ( DIF_INDEX )
          ;
          END
        ; IF STOP_TST = '  '
          THEN BEGIN
            OUT_DONE := TRUE
          ; DIF_LINE [ OUT_INDEX ] := CHR ( 13 )
          ;
          END
        ; DIF_OUT ^ := DIF_LINE [ OUT_INDEX ]
        ; PUT ( DIF_OUT )
        ; OUT_INDEX := SUCC ( OUT_INDEX )
        ; IF OUT_INDEX = MAX_ENTRY_LEN + 1
          THEN OUT_DONE := TRUE
        UNTIL OUT_DONE
      ; OUT_INDEX := 1
      ; DIF_LINE := '                              '
      ; STOP_TST := '  '
      ;
      END
```

```
  ; PROCEDURE READ_TERM ( VAR TERMINAL : TERM_LN )

(* PROCEDURE READ_TERM IS RESPONSIBLE FOR *)
(* ACCESSING AND PASSING ON TO THE OTHER  *)
(* PROCEDURES ALL USER INPUT WHICH IS     *)
(* RECIEVED VIA THE TERMINAL              *)

    ; VAR J : TERM_INDEX

    ; BEGIN
        FOR J := 1 TO MAXLN DO BEGIN TERMINAL [ J ] := ' ' END
      ; RESET ( TERM_IN )
      ; J := 1
      ; WHILE NOT ( EOLN ( TERM_IN ) )
        DO BEGIN
          TERMINAL [ J ] := TERM_IN ^
        ; J := SUCC ( J )
        ; GET ( TERM_IN )
        ;
        END
      ;
      END
```

```
  ; PROCEDURE CONVERTER ( VAR TEMP_NUM : STUB )

(* PROCEDURE CONVERTER CONVERTS CHARACTER *)
(* TYPE INPUT INTO INTEGER VALUES....THE  *)
(* INTEGER VALUES ARE USED BY PROCEDURE   *)
(* DATA_BUILDER AS CONTROL VARIABLES      *)

    ; VAR CNVTR_CNT : STUB_INDEX
      ; INT_RESULT : INTEGER
      ; SECOND_DIGIT : BOOLEAN

    ; BEGIN
      ; CNVTR_CNT := 1
      ; SECOND_DIGIT := FALSE
      ; INT_NUM := 0
      ; FOR CNVTR_CNT := 1 TO STUB_SIZE
        DO BEGIN
          IF CNVTR_CNT = 1
          THEN BEGIN
            INT_RESULT := ORD ( TEMP_NUM [ CNVTR_CNT ] ) - ORD ( '0' )
          ; INT_NUM := INT_RESULT * TEN
          ;
          END
        ;
        END
      ; IF CNVTR_CNT = 2
        THEN CASE TEMP_NUM [ CNVTR_CNT ]
          OF
            '0' , '1' , '2' , '3' , '4' , '5' , '6' , '7' , '8' , '9'
            : BEGIN
              SECOND_DIGIT := TRUE
            ; INT_RESULT
              := ORD ( TEMP_NUM [ CNVTR_CNT ] ) - ORD ( '0' )
            ;
            END
          ; OTHERWISE TEMP_NUM [ CNVTR_CNT ] := ' '
          ;
          END (* CASE TEMP_NUM OF *)
      ; IF SECOND_DIGIT
        THEN INT_NUM := INT_NUM + INT_RESULT
        ELSE INT_NUM := INT_NUM DIV TEN
      ;
      END
```

```
 ; PROCEDURE DATA_BUILDER

* PROCEDURE DATA_BUILDER CONSTRUCTS THE   *)
* DATA PART OF THE DIF FILE.  THIS IS     *)
* THE ONLY PROCEDURE WHICH CAN CREATE     *)
* INPUT DATA STRUCTURES FOR THE DIF FILE  *)
* DATA PART                               *)

   ; VAR VEC_NUM , TUP_NUM : INTEGER
     ; TERM_CNT , VCT_INDEX , TPL_CNT : INTEGER
     ; PRINTED : BOOLEAN
     ; DIF_LN : ENTRY_LN
     ; LINE_INDEX : ENTRY_INDEX
     ; TERMINAL : TERM_LN

   ; BEGIN
       PRINTED := FALSE
     ; LINE_INDEX := 1
     ; FOR LINE_INDEX := 1 TO MAX_ENTRY_LEN
       DO DIF_LN [ LINE_INDEX ] := ' '
     ; CONVERTER ( TEMP_VEC )
     ; VEC_NUM := INT_NUM
     ; CONVERTER ( TEMP_TUP )
     ; TUP_NUM := INT_NUM
     ; FOR TPL_CNT := 1 TO TUP_NUM
       DO BEGIN
         DIF_LN := DATA_1
       ; WRITE_FILE ( DIF_LN )
       ; DIF_LN := DATA_2
       ; WRITE_FILE ( DIF_LN )
       ; VCT_INDEX := 0
       ; WHILE VCT_INDEX < VEC_NUM
         DO BEGIN
         ; VCT_INDEX := SUCC ( VCT_INDEX )
         ; WRITELN
             ( 'ENTER DATA TYPE FOR TUPLE..' : 27 , TPL_CNT : 3
             , '  VECTOR..' : 10 , VCT_INDEX : 3 )
         ; IF NOT PRINTED
           THEN BEGIN
             WRITELN
           ; WRITELN
           ; WRITELN ( 'REMINDER: ENTER "0" FOR NUMERIC DATA' )
           ; WRITELN
           ; WRITELN
           ; WRITELN ( 'REMINDER: ENTER "1" FOR STRING DATA' )
           ; WRITELN
           ; WRITELN
           ; PRINTED := TRUE
           ;
           END
         ; READ_TERM ( TERMINAL )
         ; FOR LINE_INDEX := 1 TO MAX_ENTRY_LEN
           DO BEGIN
             DIF_LN [ LINE_INDEX ] := TERMINAL [ LINE_INDEX ]
           ;
           END
```

```
    ; CASE DIF_LN [ 1 ]
      OF
        '0'
        : BEGIN
          DIF_LN := DATA_3
        ; WRITELN ( 'ENTER NUMERIC DATA' )
        ; READ_TERM ( TERMINAL )
        ; TERM_CNT := 1
        ; FOR LINE_INDEX := 3 TO MAX_ENTRY_LEN
          DO BEGIN
            DIF_LN [ LINE_INDEX ] := TERMINAL [ TERM_CNT ]
          ; TERM_CNT := SUCC ( TERM_CNT )
          END
        ; WRITE_FILE ( DIF_LN )
        ; DIF_LN := DATA_4
        ; WRITE_FILE ( DIF_LN )
        ;
        END
      ; '1'
        : BEGIN
          DIF_LN := DATA_5
        ; WRITE_FILE ( DIF_LN )
        ; WRITELN ( 'ENTER STRING DATA' )
        ; READ_TERM ( TERMINAL )
        ; TERM_CNT := 1
        ; FOR LINE_INDEX := 1 TO MAX_ENTRY_LEN
          DO BEGIN
          ; DIF_LN [ LINE_INDEX ] := TERMINAL [ TERM_CNT ]
          ; TERM_CNT := SUCC ( TERM_CNT )
          END
        ; WRITE_FILE ( DIF_LN )
        ;
        END
      ; OTHERWISE
        BEGIN
        ; WRITELN ( 'ATTEMPT TO ENTER INVALID DATA TYPE' )
        ; WRITELN ( 'RE-ENTER DATA TYPE AS INSTRUCTED' )
        ; VCT_INDEX := PRED ( VCT_INDEX )
        ;
        END
      END (* CASE DIF_LN OF *)
    ;
    END
  ;
  END
; DIF_LN := DATA_1
; WRITE_FILE ( DIF_LN )
; DIF_LN := DATA_6
; WRITE_FILE ( DIF_LN )
;
END
```

```
  ; PROCEDURE HEAD_BUILDER

(* THIS PROCEDURE CONSTRUCTS THE HEADER    *)
(* OF THE DIF FILE.  IT IS WITHIN THIS     *)
(* PROCEDURE THAT THE NUMBER OF VECTORS    *)
(* AND TUPLES THE FILE WILL CONTAIN IS     *)
(* ESTABLISHED BY THE USER.                *)

    ; VAR VEC_CNT , HEAD_CNT , TEMP_CNT , TUP_CNT : ENTRY_INDEX
      ; LOOP_CNT , VEC_NUM , TUP_NUM : INTEGER
      ; HEAD_DONE , COUNT_DONE : BOOLEAN
      ; TERMINAL : TERM_LN
      ; FILE_LN : ENTRY_LN

    ; BEGIN
        WRITELN
          (
  'THIS PROCEDURE CONSTRUCTS A STANDARD HEADING SECTION FOR A DIF_FILE'
          )
      ; WRITELN ( 'ENTER THE NUMBER OF VECTORS' )
      ; READ_TERM ( TERMINAL )
      ; VEC_CNT := 1
      ; HEAD_CNT := 1
      ; TEMP_CNT := 1
      ; TUP_CNT := 1
      ; FOR HEAD_CNT := 1 TO MAX_ENTRY_LEN
        DO FILE_LN [ HEAD_CNT ] := ' '
      ; HEAD_CNT := 1
      ; FOR VEC_CNT := 1 TO 2
        DO BEGIN
          CASE TERMINAL [ HEAD_CNT ]
          OF
            '0' , '1' , '2' , '3' , '4' , '5' , '6' , '7' , '8' , '9'
            : BEGIN
              TEMP_VEC [ HEAD_CNT ] := TERMINAL [ HEAD_CNT ]
            ; HEAD_CNT := SUCC ( HEAD_CNT )
            ;
            END
          ; OTHERWISE TEMP_VEC [ HEAD_CNT ] := ' '
          ;
          END
        ;
        END
```

```
; WRITELN ( 'ENTER THE NUMBER OF TUPLES' )
; READ_TERM ( TERMINAL )
; TUP_CNT := 1
; HEAD_CNT := 1
; FOR TUP_CNT := 1 TO 2
  DO BEGIN
    CASE TERMINAL [ HEAD_CNT ]
    OF
      '0' , '1' , '2' , '3' , '4' , '5' , '6' , '7' , '8' , '9'
      : BEGIN
        TEMP_TUP [ HEAD_CNT ] := TERMINAL [ HEAD_CNT ]
      ; HEAD_CNT := SUCC ( HEAD_CNT )
      ;
      END
    ; OTHERWISE TEMP_TUP [ HEAD_CNT ] := ' '
    ;
    END (* CASE OF TEMP_TUP *)
  ;
  END (* FOR TUP_CNT *)
; HEAD_CNT := 1
; VEC_CNT := 1
; TUP_CNT := 1
; LOOP_CNT := 0
; REPEAT BEGIN
      LOOP_CNT := SUCC ( LOOP_CNT )
    ; CASE LOOP_CNT
      OF
        1
        : BEGIN
          FILE_LN := HEAD_1
        ; WRITE_FILE ( FILE_LN )
        ; FILE_LN := HEAD_5
        ; WRITE_FILE ( FILE_LN )
        ;
        END (* FIRST HEADING ITEM *)
      ; 2
        : BEGIN
          FILE_LN := HEAD_2
        ; WRITE_FILE ( FILE_LN )
        ; FILE_LN := HEAD_5
        ; FOR HEAD_CNT := 3 TO 4
          DO BEGIN
            FILE_LN [ HEAD_CNT ] := TEMP_VEC [ VEC_CNT ]
          ; VEC_CNT := SUCC ( VEC_CNT )
          ;
          END
        ; WRITE_FILE ( FILE_LN )
        ;
        END (* SECOND HEADING ITEM *)
      ; 3
        : BEGIN
          FILE_LN := HEAD_3
        ; WRITE_FILE ( FILE_LN )
        ; FILE_LN := HEAD_5
        ; FOR HEAD_CNT := 3 TO 4
          DO BEGIN
            FILE_LN [ HEAD_CNT ] := TEMP_TUP [ TUP_CNT ]
          ; TUP_CNT := SUCC ( TUP_CNT )
          ;
          END
```

```
        ; WRITE_FILE ( FILE_LN )
        ;
        END (* THIRD HEADING ITEM *)
      ; 4
        : BEGIN
          FILE_LN := HEAD_4
        ; WRITE_FILE ( FILE_LN )
        ; FILE_LN := HEAD_6
        ; WRITE_FILE ( FILE_LN )
        ;
        END (* FORTH HEADING ITEM *)
      ; OTHERWISE HEAD_DONE := TRUE
      END (* CASE LOOP_CNT OF *)
    ; FILE_LN := HEAD_Q
    ; WRITE_FILE ( FILE_LN )
    ; IF LOOP_CNT = ITEM_LIMIT
      THEN HEAD_DONE := TRUE
    ;
    END
  ;
  UNTIL HEAD_DONE
; DONE_FLAG := FALSE
;
END
```

```
  ; PROCEDURE INIT_FILE

(* INIT_FILE ESTABLISHES THE INTERACTIVE  *)
(* RELATIONSHIP WITH THE USER.            *)

    ; VAR TERM_MSG : TERM_LN
      ; TEST : ARRAY [ 1 .. 5 ] OF CHAR
      ; LN_INDEX : TERM_INDEX

    ; BEGIN
      ; DONE_FLAG := FALSE
      ; WRITELN
          (
           'THIS PROGRAM BUILDS A "DATA INTERCHANGE FORMAT" (DIF) FILE.'
          )
      ; WRITELN
      ; WRITELN
      ; WRITELN
          (
          'DURING THIS SESSION A DIF FILE WILL BE CREATED AND WRITTEN'
          )
      ; WRITELN ( 'INTO YOUR FILE SPACE' )
      ; WRITELN
      ; WRITELN
      ; WRITELN ( 'THE FILE CAN BE ACCESSED BY REFERING TO DIF.PE' )
      ; WRITELN
      ; WRITELN
      ; WRITELN
          (
         'FOR FURTHER INFORMATION REFER TO "THE DIF BUILDER TUTORIAL"'
          )
      ; WRITELN ( 'APPENDIX "B" TO KSU MASTER"S REPORT ENTITLED :' )
      ; WRITELN ( '   "DATA INTERCHANGE FORMAT FILES:' )
      ; WRITELN ( 'A SIMPLE, DIRECT APPROACH TO PROVIDING' )
      ; WRITELN ( '     TRANSPROTABLE GRAPHICS DATA"' )
      ; WRITELN
      ; WRITELN
      ; WRITELN ( 'WHEN YOU ARE READY TO CONTINUE TYPE "START"' )
      ; WRITELN ( 'TO TERMINATE TYPE "STOP"' )
      ; READ_TERM ( TERM_MSG )
      ; FOR LN_INDEX := 1 TO 5
        DO BEGIN
        ; TEST [ LN_INDEX ] := TERM_MSG [ LN_INDEX ]
        ;
        END
      ; IF TEST = 'START'
        THEN HEAD_BUILDER
        ELSE BEGIN
        ; WRITELN ( 'PROGRAM TERMINATED' )
        ; DONE_FLAG := TRUE
        ;
        END
      END
```

```
; BEGIN
  ; REWRITE ( DIF_OUT )
  ; TEMP_VEC := '  '
  ; TEMP_TUP := '  '
  ; INIT_FILE
  ; IF DONE_FLAG = FALSE
    THEN DATA_BUILDER
    ELSE
  END
```

```
XDEL DIF.PE
AL DIF.PE,IN,80
LO DIFBLDR
AS 0,CON:
AS 1,CON:
AS 2,DIF.PE
ST
$EXIT
```

A2

DIF.PURGE PROGRAM LISTING

```
5     REM  PROGRAM DIF.PURGE

10    CLEAR : REM  SETS ALL FIELDS TO ZERO

20    PRINT D$;"MAXFILES 5": PRINT

22    REM  VARIABLE DECLARATION SECTION

24    REM  A$ - ARRAY IDENTIFIER

25    REM  B$ - VARIABLE USED TO RETRIEVE
      INPUT CHARACTERS

26    REM  C$ - USED TO TEST FOR THE END
      OF THE FILE

30    REM  Z$ - INPUT FILE IDENTIFIER

40    REM  W$ - OUTPUT FILE IDENTIFIER

45    REM  "I" - INPUT LOOP-CONTROL VARIABLE..
      USED TO IDENTIFY EACH ROW IN AN ARRAY

50    REM  "J" - INPUT/OUTPUT LOOP-CONTROL
      VARIABLE..USED TO IDENTIFY EACH CHARACTER
      IN EACH ROW OF AN ARRAY

55    REM  "K" - OUTPUT LOOP-CONTROL VARIABLE..
      USED TO IDENTIFY THE CURRENT ROW DURING THE
      OUTPUT PROCESS

70    D$ =  CHR$ (4): REM  CONTROL D (ET)

90    R$ =  CHR$ (13): REM  CONTROL M (CR)

100   U$ =  CHR$ (95): REM  (UNDERSCORE)

110   DIM A$(200,20): REM  DEFINES AN INPUT
      OUTPUT ARRAY

120   PRINT "THIS PROGRAM RETRIEVES AND EDITS
      TEXT FILES CREATED BY THE DATA CAPTURE PROGRAM"

130   PRINT

140   PRINT "INPUT FILES ARE ACCESSED FROM
      DISK 1"

141   PRINT

150   PRINT "OUTPUT FILES ARE WRITTEN TO
      DISK 2"

151   PRINT
```

```
160   PRINT "MON C,I,O IS IN EFFECT"

170   PRINT : PRINT : REM  DOUBLE SPACE

180   INPUT "ENTER INPUT FILENAME ";Z$: PRINT :
      REM  REQUIRES THE USER TO IDENTIFY THE
      INPUT FILE

190   INPUT "ENTER OUTPUT FILENAME ";W$: PRINT :
      PRINT : REM  ALLOWS USER TO NAME HIS
      OUTPUT FILE

210   PRINT D$;"MON C,I,O": PRINT :
      REM  ALLOWS USER TO MONITOR DOS (DISK
      OPERATING SYSTEM ) COMMANDS AND DOS I/O

220   PRINT D$;"OPEN ";Z$;",D1": REM  OPENS
      THE FILE TO BE REFORMATTED

230   PRINT D$;"READ  ";Z$: REM  READS THE
      FILE TO BE REFORMATTED
240   I = I + 1: REM  COUNTER FOR ROWS IN THE
      ARRAY

250   J = 0: REM  COUNTER FOR CHARACTERS IN
      EACH ROW

260   GET B$: REM  RETRIEVES A SINGLE CHARACTER
      FROM THE INPUT FILE

270   FLAG = 0: REM  INITIALIZES "FLAG"..
      "0"INDICATES THAT THE CURRENT CHARACTER
      IS VALID

280   IF B$ = R$ THEN  GOSUB 1000:
      REM  THIS IS AN END OF FILE TEST

300   IF B$ = "-" THEN  GOSUB 2000:
      REM  TESTS FOR A VALID DASH OR MINUS
      SIGN

305   IF FLAG = 1 THEN  GOTO 260: REM
      CAUSES THE NEXT CHARACTER TO BE RETRIEVED
      WHEN THE DASH IS NOT VALID

310   IF B$ = U$ THEN  GOTO 260: REM  DELETES
      THE UNDERSCORE THEN GETS NEXT CHAR.

320   J = J + 1: REM  NUMBER OF CHARACTERS
      IN THE CURRENT ROW IS INCREMENTED
      BY ONE
```

```
330   A$(I,J) = B$: REM  ASSIGNS THE CURRENT
      INPUT CHARACTER TO THE INPUT ARRAY

337   C$ = B$: PRINT B$;: REM  C$ IS USED AT
      LINE 1000 TO TEST FOR END OF DATA

340   IF B$ = R$ GOTO 240: REM  TESTS FOR A
      CARRIAGE RETURN..IF TRUE, "I" IS INCREMENTED
      BY ONE AND "J" IS SET TO ZERO

350   GOTO 260: REM  LOOPS BACK TO ACCESS
      NEXT CHARACTER

360   REM  THE FOLLOWING SECTION IS USED
      TO WRITE THE REFORMATTED FILE TO
      DISK 2. IF A FILE OF THE SAME NAME ALREADY
      EXISTS ON DISK 2 IT WILL BE OVERWRITTEN

400   PRINT R$;D$"OPEN";W$;",D2

410   PRINT D$;"DELETE ";W$

420   PRINT D$;"OPEN ";W$;",D2

430   PRINT D$;"WRITE ";W$

440   I = I - 1: REM  RESETS "I" EQUAL TO
      THE LAST ROW IN THE INPUT ARRAY WHICH
      HAD VALID INPUT

445   J = 0

450   FOR K = 1 TO I: REM  "K" IS ESTABLISHED
      AS A LOOP CONTROL VARIABLE..WHEN "K" IS
      GREATER THAN "I" THEN THE ARRAY WILL BE
      EMPTY

460   J = 1: REM  SETS "J" TO THE FIRST CHARACTER
      IN EACH ROW

470   FOR J = 1 TO 80

480   PRINT A$(K,J);

490   IF A$(K,J) <  > R$ THEN  NEXT J: REM
      TEST FOR END OF CURRENT OUTPUT LINE

500   NEXT K

600   PRINT D$;"CLOSE ";W$: REM  FILE 'HOUSEKEEPING'

610   PRINT D$;"CLOSE ";Z$: REM  FILE 'HOUSEKEEPING
```

```
620   PRINT D$;"NOMON C,I,O": REM  TURN MONITOR OFF

630   END

900   REM  THIS SUBROUTINE TESTS FOR THE
      END OF THE INPUT FILE

1000  IF B$ = C$ THEN  GOTO 400: REM  THIS
      CONDITION CAN ONLY OCCUR IF THE END OF
      THE INPUT FILE HAS BEEN REACHED..IF TRUE,
      CONTROL IS SENT TO THE PROGRAM OUTPUT
      SECTION.

1010  RETURN

2000  REM  TEST FOR VALID DASH OR MINUS SIGN

2010  IF H = I THEN FLAG = 0: REM  INDICATES A
      DASH HAS ALREADY BEEN REMOVED FROM THE
      CURRENT INPUT LINE

2020  IF H <  > I THEN FLAG = 1: REM
      INDICATES A DASH HAS NOT YET BEEN REMOVED
      FROM THE CURRENT LINE

2030  IF H <  > I THEN H = I: REM  SETS
      "H" TO THE CURRENT INPUT LINE VALUE..
      THIS IS USED TO INDICATE A DASH HAS
      BEEN REMOVED FROM THE CURRENT LINE

2040  RETURN
```

APPENDIX B

DIF BUILDER TUTORIAL

INTRODUCTION

The DIF BUILDER TUTORIAL consists of step-by-step instructions for implementing the DIF_BUILDER SYSTEM. The system is designed to allow the user to create a DIF file on the Perkin-Elmer 8/32 computer and then transfer the file to an Apple II computer. The DIF file serves as a medium by which the user can transport data. Once transfered the DIF file is used to provide a data input stream to the Visiplot/Visitrend graphics plotting package. To successfully create and transfer a DIF file four separate phases of activity must be accomplished. Each of the phases will be discussed in detail in the following four sections

SECTION I

PHASE I: FILE CONSTRUCTION

Phase I is the only phase conducted on the Perkin-Elmer 8/32 computer. Implementation of this phase requires the DIF_BUILDER program and the BUILDER command substitution system (CSS) file to be resident on the host computer. These files may be either in a public access mode or in the user's file space.

STEP I: You should organize your data in TUPLE, VECTOR order. This can be thought of as row, column order, with the TUPLEs equating to rows and the columns equating to VECTORs. Each TUPLE can be viewed as containing data from a 'snapshot' in time. The VECTORs would represent the components of the 'snapshot.' For example, a file containing revenue, cost and profit data for an entire year could be visualized as a data matrix. Each month of the year would be assigned a TUPLE (or row) and each TUPLE would contain three VECTORs (or columns). In this case the VECTORs in each TUPLE would be revenue, cost and profit for a given month. For a more detailed explaination of VECTORs and TUPLEs and how they are used to organize data see Chapter III of this report.

STEP II: Once the data is organized, sign-on to the host computer. After signing-on, activate DIF_BUILDER by entering "BUILDER". This produces a call to a CSS file which activates the DIF_BUILDER program.

STEP III: When the DIF_BUILDER program is initiated the screen will display the following messages:

```
"THIS PROGRAM BUILDS A DATA INTERCHANGE FORMAT FILE"

"DURING THIS SESSION A DIF FILE WILL BE CREATED AND
 WRITTEN INTO YOUR FILE SPACE"

"THE FILE CAN BE ACCESSED BY REFERING TO DIF.PE"

"FOR FURTHER INFORMATION REFER TO THE DIF BUILDER
 TUTORIAL"
"APPENDIX 'B' TO KSU MASTER'S REPORT ENTITLED:"
"    DATA INTERCHANGE FORMAT FILES    "
"A SIMPLE DIRECT APPROACH TO PROVIDING"
"    TRANSPORTABLE GRAPHICS DATA      "

"WHEN YOU ARE READY TO CONTINUE TYPE 'START'"

"TO TERMINATE TYPE 'STOP'"
```

If you respond with input other than "START" the message "PROGRAM TERMINATED" will appear and the program will terminate. In order to continue, enter "START". The following messages will appear:

```
"THIS PROCEDURE CREATES A STANDARD HEADING SECTION
 FOR A DIF FILE"

"ENTER THE NUMBER OF VECTORS"
```

After you have entered the number of VECTORs in your

data matrix you will recieve the following message:

"ENTER THE NUMBER OF TUPLES"

Respond to this message by entering the number of TUPLEs contained in your data matrix. Next you will be prompted to enter the data TYPE INDICATOR for each VECTOR within each TUPLE.

"ENTER DATA TYPE FOR TUPLE..1 VECTOR..1"

The following reminders occur only on the first iteration of the data entry loop.

"REMINDER: ENTER '0' FOR NUMERIC DATA"

"REMINDER: ENTER '1' FOR STRING DATA"

At this point enter the data TYPE INDICATOR for the TUPLE and VECTOR as indicated in the prompt. The next prompt will depend upon the data TYPE INDICATOR most recently entered.

"ENTER NUMERIC DATA"

or

"ENTER STRING DATA"

When the prompt is recieved, enter the DATA VALUE or STRING VALUE as indicated in the prompt. The program will continue to request data until input has been acquired for each VECTOR of each TUPLE.

The program will terminate once all items within the data matrix have been satisfied. At this point a file entitled DIF.PE will have been written into your file space. DIF.PE can be treated as any other file in your file space. If you would like to rename DIF.PE now would be a good time to do so. If you intend to create more than one DIF file, DIF.PE will have to be renamed since DIF_BUILDER over-writes any existing file named "DIF.PE" each time it is run. This ends PHASE I.

PHASE II: FILE TRANSFER

During PHASE II you will transfer a file from the host computer to the Apple II computer. To do this you will need to use two of the four diskettes which help to make up the DIF BUILDER SYSTEM. The diskettes are labeled "VISIPLOT/VISITREND", "DIF.PURGE", "DATA CAPTURE 4.0" and "PURGED FILES." Place the diskette labeled "DATA CAPTURE 4.0" into DRIVE# 1 and the diskette labeled "DIF.PURGE" into DRIVE# 2 (see Figure B.1). Switch the Apple II and the monitor "ON" after inserting the diskettes. The "ON/OFF" switch for the Apple II is located at the left-rear of the machine.

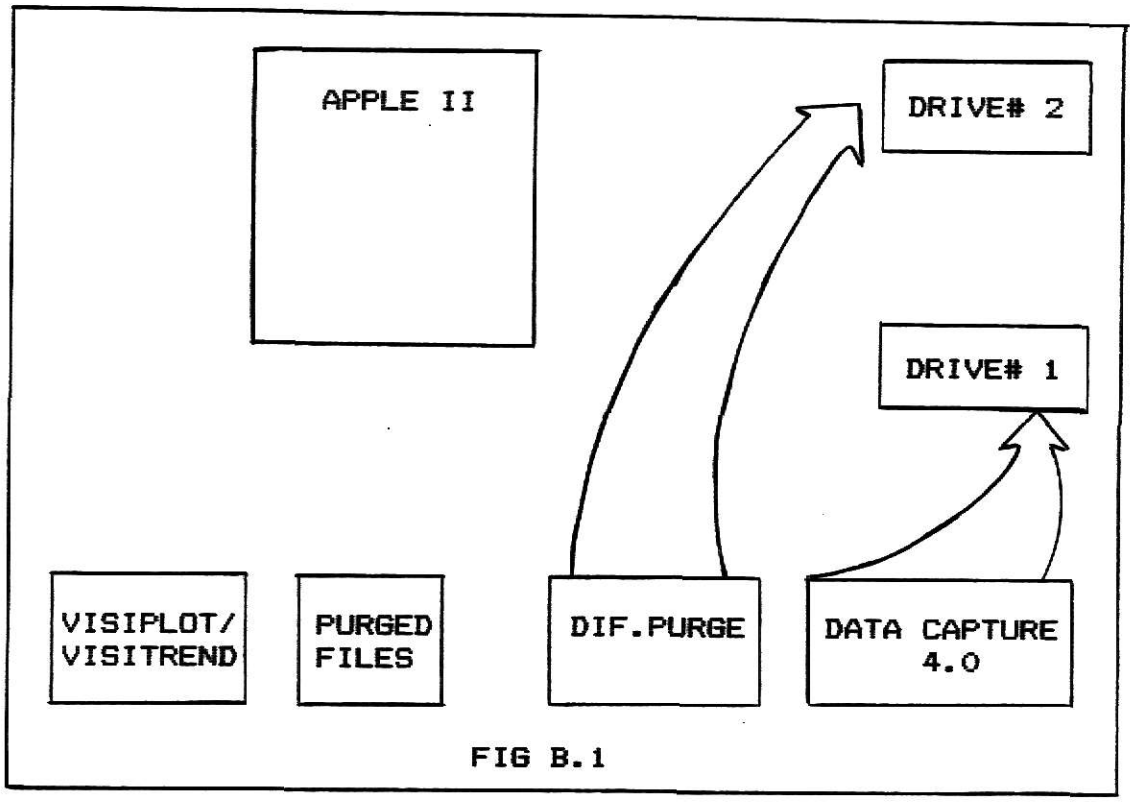


FIG B.1

After a short pause the following heading will appear on your monitor:

DRIVE = 1 CAPTURE ON TRANSMIT ON

LINES = 0 SPECHAR OFF DUPLEX HALF

BAUD = 300 REMOTE CARRIER OFF

* INDICATES INVERSE DISPLAY:

FIG B.2

The following series of steps changes the DATA CAPTURE 4.0 default conditions, thus making it compatible with the host

computer.

1. PRESS "ESC" (accesses a generalized menu)
2. PRESS "T" (accesses a "toggle" menu"
3. PRESS "A" (establishes DRIVE# 2 as the primary drive)

   RESULT: DRIVE = 2
4. PRESS "ESC"
5. PRESS "T"
6. PRESS "C" (disables data capture ability)

   RESULT: CAPTURE OFF
7. PRESS "ESC"
8. PRESS "T"
9. PRESS "D" (puts data capture in duplex mode)

   RESULT: DUPLEX FULL

Data capture 4.0 is now properly configured - your monitor should appear as follows:

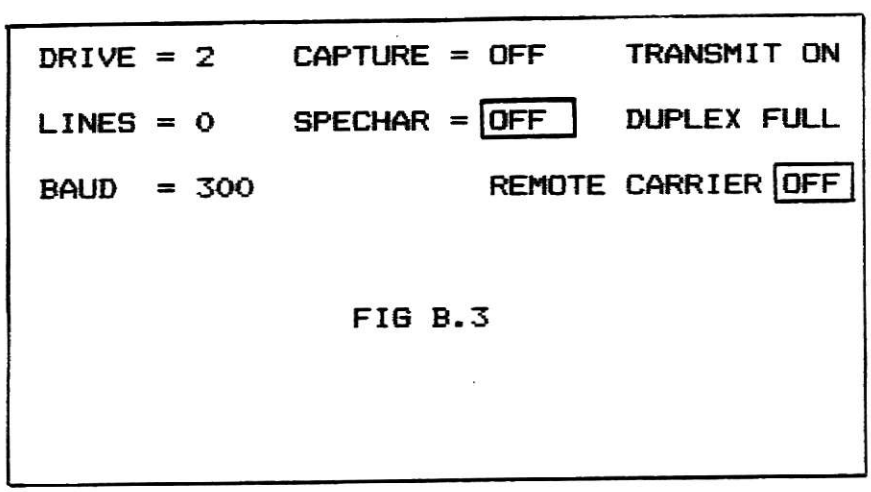


FIG B.3

To establish a communications link with the host computer follow these steps:

1. PRESS "ESC"
2. PRESS "E" (This allows you to enter the phone number

through which you will be linked to the host computed.)

MONITOR DISPLAY:

PHONE DIALING ROUTINE

ENTER THE NUMBER TO BE DIALED

PRESS RETURN TO ABORT

PHONE# _____

(NOTE: Current numbers through wihich the Perkin-Elmer can be accessed are: 537-0463/1832. The dash is optional.)

At this point you should be linked to the host computer. The Apple II will be treated as any of the other terminals hooked to the host, except it will be slower (300 baud). Take the following steps:

1. SIGN-ON AS YOU NORMALLY DO
2. TYPE "COPYP filename,CON: [DO NOT PRESS RETURN]
3. PRESS "ESC" (MENU..You are in the "smart" terminal mode)
4. PRESS "T" ("toggle" mode)
5. PRESS "C" (enables data capture)

   RESULT: CAPTURE ON
6. PRESS "RETURN"(Wait for a few seconds)

The DIF file will appear on the monitor as it is being loaded into memory. Note that the maximum number of lines which can be transfered via DATA CAPTURE 4.0 is 500. To

transfer larger files see the DATA CAPTURE documentation. When the entire file has been transfered, review your file by using the DATA CAPTURE list feature. But first you must disable the data capture feature. Take the following steps:

1. PRESS "ESC"
2. PRESS "T"
3. PRESS "C"
4. PRESS "ESC"
5. PRESS "L"
6. PRESS "RETURN"
7. PRESS <any key> (to stop listing)
8. PRESS "ESC" (to continue listing)

Once satisfied that the file is complete, you must remove any excess lines which were transfered as part of the data capture process. Excess lines will appear prior to the first line of the DIF file and after the last legitimate line in the DIF file. The first and last lines of DIF file should always appear as shown below:

FIRST LINE ..... ___-__TABLE

LAST LINE ..... ___-__

In Figure B.4, lines 1 through 3 would be have to be deleted. To delete unwanted lines:

1. PRESS "ESC"

2. PRESS "D"
3. ENTER BEGINING LINE# (line# 1 in FIG B.4)
4. PRESS "RETURN"
5. ENTER ENDING LINE# (line# 3 in FIG B.4)
6. PRESS "RETURN"
7. RE-LIST THE FILE (Note that the line numbers have changed.)
8. IDENTIFY UNWANTED LINES AT THE END OF THE FILE (Lines 63 and 64 in FIG B.5)
9. PRESS "ESC"
10. PRESS "D"
11. ENTER LINE NUMBERS TO BE DELETED
12. PRESS "RETURN"

After completion your file should appear as in Figure B.6. Note the first and last lines. Once satisfied with the transfered file write the file to the diskette. Proceed as indicated:

1. CHECK MONITOR DISPLAY TO INSURE THAT YOU ARE WRITING TO THE CORRECT DRIVE (I.E. DRIVE = 2). IF NOT, PRESS "ESC", PRESS "T", PRESS "A". (Now your display should read: DRIVE = 2.)
2. PRESS "ESC"
3. PRESS "W"
4. ENTER THE FILENAME TO BE SAVED. (You can rename your file at this point if you so desire.)
5. PRESS "RETURN" (Your file is now being saved to the

```
1
2.
3.   _
4.   ___-__TABLE_
5.   ___-__0,1_
6.   ___-__""_
7.   ___-__VECTORS_
8.   ___-__0,1_
9.   ___-__""_
10.  ___-__TUPLES_
11.  ___-__0,12_
12.  ___-__""_
13.  ___-__DATA_
14.  ___-__0,0_
15.  ___-__""_
16.  ___-__-1,0_
17.  ___-__BOT_
18.  ___-__0,103_
19.  ___-__V_
20.  ___-__-1,0_
21.  ___-__BOT_
22.  ___-__0,107_
23.  ___-__V_
24.  ___-__-1,0_
25.  ___-__BOT_
26.  ___-__0,119_
27.  ___-__V_
28.  ___-__-1,0_
29.  ___-__BOT_
30.  ___-__0,105_
31.  ___-__V_
32.  ___-__-1,0_
33.  ___-__BOT_
34.  ___-__0,87_
35.  ___-__V_
36.  ___-__-1,0_
37.  ___-__BOT_
38.  ___-__0,82_
39.  ___-__V_
40.  ___-__-1,0_
41.  ___-__BOT_
42.  ___-__0,79_
43.  ___-__V_
44.  ___-__-1,0_
45.  ___-__BOT_
46.  ___-__0,93_
47.  ___-__V_
```

FIG B.4

(CONTINUES ON NEXT PAGE)

```
48. ___-__-1,0_
49. ___-__BOT_
50. ___-__0,62_
51. ___-__V_
52. ___-__-1,0_
53. ___-__BOT_
54. ___-__0,54_
55. ___-__V_
56. ___-__-1,0_
57. ___-__BOT_
58. ___-__0,86_
59. ___-__V_
60. ___-___-1,0_
61. ___-__BOT_
62. ___-__0,87_
63. ___-__V_
64. ___-__-1,0_
65. ___-__EOD_
66. ___-___
67. ___-__JIM      --ND OF TASK CODE=
0_
68.___ELAPSED TIME=00:00:32_
```

FIG B.4(CONT.)

```
1.  ___-__TABLE_
2.  ___-__0,1_
3.  ___-__""_
4.  ___-__VECTORS_
5.  ___-__0,1_
6.  ___-__""_
7.  ___-__TUPLES_
8.  ___-__0,12_
9.  ___-__""_
10. ___-__DATA_
11. ___-__0,0_
12. ___-__""_
13. ___-__-1,0_
14. ___-__BOT_
15. ___-__0,103_
16. ___-__V_
17. ___-__-1,0_
18. ___-__BOT_
19. ___-__0,107_
20. ___-__V_
21. ___-__-1,0_
22. ___-__BOT_
23. ___-__0,119_
24. ___-__V_
25. ___-__-1,0_
26. ___-__BOT_
27. ___-__0,105_
28. ___-__V_
29. ___-__-1,0_
30. ___-__BOT_
31. ___-__0,87_
32. ___-__V_
33. ___-__-1,0_
34. ___-__BOT_
35. ___-__0,82_
36. ___-__V_
37. ___-__-1,0_
38. ___-__BOT_
39. ___-__0,79_
40. ___-__V_
41. ___-__-1,0_
42. ___-__BOT_
42. ___-__0,93_
43. ___-__V_
```

FIG B.5

(CONTINUES ON NEXT PAGE)

```
44. ___-__-1,0_
45. ___-__BOT_
46. ___-__0,62_
47. ___-__V_
48. ___-__-1,0_
49. ___-__BOT_
50. ___-__0,54_
51. ___-__V_
52. ___-__-1,0_
53. ___-__BOT_
54. ___-__0,86_
55. ___-__V_
56. ___-___-1,0_
57. ___-__BOT_
58. ___-__0,87_
59. ___-__V_
60. ___-__-1,0_
61. ___-__EOD_
62. ___-___
63. ___-__JIM      --ND OF TASK CODE=
0_
64. ___ELAPSED TIME=00:00:32_
```

FIG B.5(CONT.)

```
1.  ___-__TABLE_
2.  ___-__0,1_
3.  ___-__""_
4.  ___-__VECTORS_
5.  ___-__0,1_
6.  ___-__""_
7.  ___-__TUPLES_
8.  ___-__0,12_
9.  ___-__""_
10. ___-__DATA_
11. ___-__0,0_
12. ___-__""_
13. ___-__-1,0_
14. ___-__BOT_
15. ___-__0,103_
16. ___-__V_
17. ___-__-1,0_
18. ___-__BOT_
19. ___-__0,107_
20. ___-__V_
21. ___-__-1,0_
22. ___-__BOT_
23. ___-__0,119_
24. ___-__V_
25. ___-__-1,0_
26. ___-__BOT_
27. ___-__0,105_
28. ___-__V_
29. ___-__-1,0_
30. ___-__BOT_
31. ___-__0,87_
32. ___-__V_
33. ___-__-1,0_
34. ___-__BOT_
35. ___-__0,82_
36. ___-__V_
37. ___-__-1,0_
38. ___-__BOT_
39. ___-__0,79_
40. ___-__V_
41. ___-__-1,0_
42. ___-__BOT_
42. ___-__0,93_
43. ___-__V_
```

FIG B.6

(CONTINUES ON NEXT PAGE)

```
44. ___-__-1,0_
45. ___-__BOT_
46. ___-__0,62_
47. ___-__V_
48. ___-__-1,0_
49. ___-__BOT_
50. ___-__0,54_
51. ___-__V_
52. ___-__-1,0_
53. ___-__BOT_
54. ___-__0,86_
55. ___-__V_
56. ___-___-1,0_
57. ___-__BOT_
58. ___-__0,87_
59. ___-__V_
60. ___-__-1,0_
61. ___-__EOD_
62. ___-___
```

NOTE: LINE NUMBERS ARE PROVIDED BY THE DATA CAPTURE PROGRAM AND WILL NOT BE WRITTEN INTO YOUR FILE.

FIG B.6(CONT.)

diskette as a text file.)

Once the file has been written onto the diskette, PRESS "RETURN". This will cause the host computer prompt to appear. You can now sign-off the host computer. After signing-off, sever the communications link between the two computers as follows:

1. PRESS "ESC"
2. PRESS "H" (Severs the phone link between the computers)
3. PRESS "Q" (Exits the DATA CAPTURE 4.0 program.)
4. PRESS "Y"
5. PRESS "RETURN"

This completes phase II.

PHASE III: REFORMATING THE FILE

During PHASE III, the file transfered during PHASE II will be reformated so as to make it compatitble with the Apple II computer. To reformat a file, take the following steps:

1. PLACE THE "DIF.PURGE" DISKETTE IN DRIVE# 1 (see FIG B.7)
2. PLACE THE "PURGED FILES" DISKETTE IN DRIVE# 2

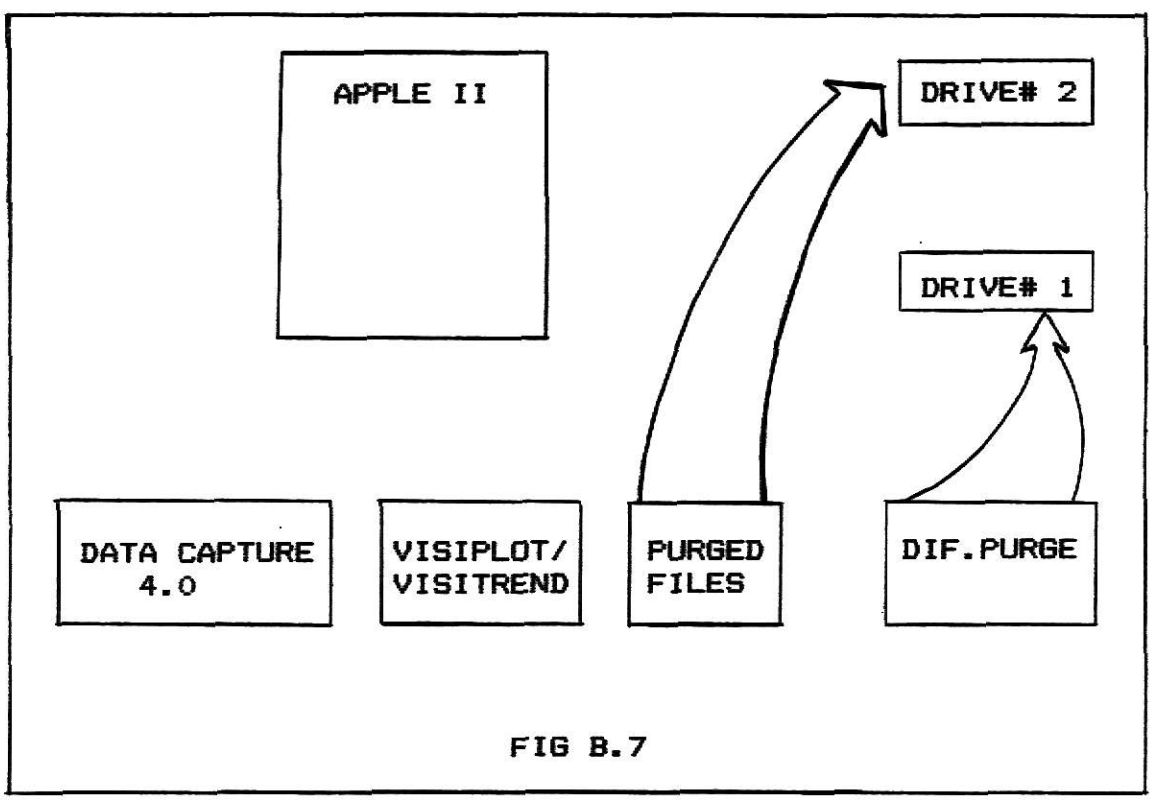


FIG B.7

3. ENTER "PR#6" (NOTE: All entries are followed by a

carriage return!)

4. ENTER "CATALOG" (A listing of the files contained on the diskette will be displayed. Among those listed should be the name of the file you transfered during PHASE II, in addition there should alwo be a file named DIF.PURGE. If the file you wish to reformat is not on the diskette in DRIVE# 1, proceed as follows: ENTER "LOAD DIF.PURGE"; put the diskette with the file to be reformated in DRIVE# 1; ENTER "RUN". Proceed to step# 5.
5. ENTER "RUN DIF.PURGE"
6. ENTER <filename> (This is the file which you wish to format.)
7. ENTER <filename> (This is the name the file will have after it has been reformated. This can be the same as the input filename. However, it is suggested that a different name be used for the output file.)

After entering the output filename DIF.PURGE will reformat the input file. Your monitor will display the file as it is written into thé computer's memory and then again as it is written onto the "PURGED FILES" diskette in DRIVE# 2. You now have a useable DIF file on the "PURGED FILES" diskette. Figure B.8 depicts the typical structure of this file. This concludes PHASE III.

```
TABLE
0,1
""
VECTORS
0,1
""
TUPLES
0,12
""
DATA
0,0
""
-1,0
BOT
0,103
V
-1,0
BOT
0,107
V
-1,0
BOT
0,119
V
-1,0
BOT
0,105
V
-1,0
BOT
0,87
V
-1,0
BOT
0,82
V
-1,0
BOT
0,79
V
-1,0
BOT
0,93
V
```

FIG B.8

(CONTINUES ON NEXT PAGE)

```
-1,0
BOT
0,62
V
-1,0
BOT
0,54
V
-1,0
BOT
0,86
V
-1,0
BOT
0,87
V
-1,0
EOD
```

FIG B.8(CONT.)

SECTION IV

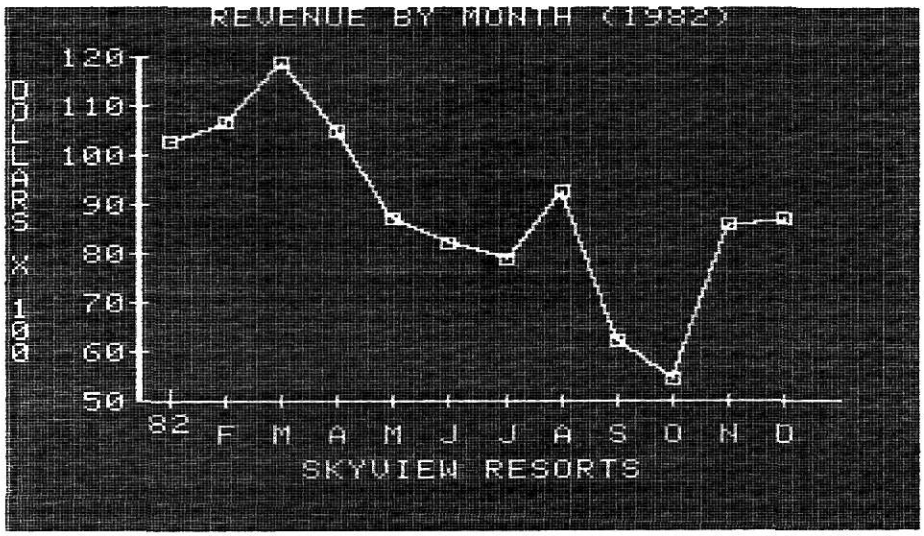
PHASE IV: DISPLAYING THE FILE

During PHASE IV the file which was reformated during PHASE III will be graphicaly displayed by employing the VISIPLOT/VISITREND shoftware package. Take the following steps:

1. INSERT THE VISIPLOT/VISITREND DISKETTE IN DRIVE# 1
2. INSERT THE DISKETTE LABELED "PURGED FILES" INTO DRIVE# 2
3. ENTER "PR#6" (This will load VISIPLOT/VISITREND into the Apple II.)
   (NOTE: For a comprehensive tutorial on VISIPLOT/VISITREND refer to the "VISICALC/VISIPLOT" LESSON MANUAL" by Donald L. Loftus, Graduate Student, Kansas State University and to the VISIPLOT/VISITREND documentation provided with the software package.)
4. TO LOAD YOUR FILE FOLLOW THE INSTRUCTIONS PROVIDED BY THE VISIPLOT/VISITREND PROGRAM.

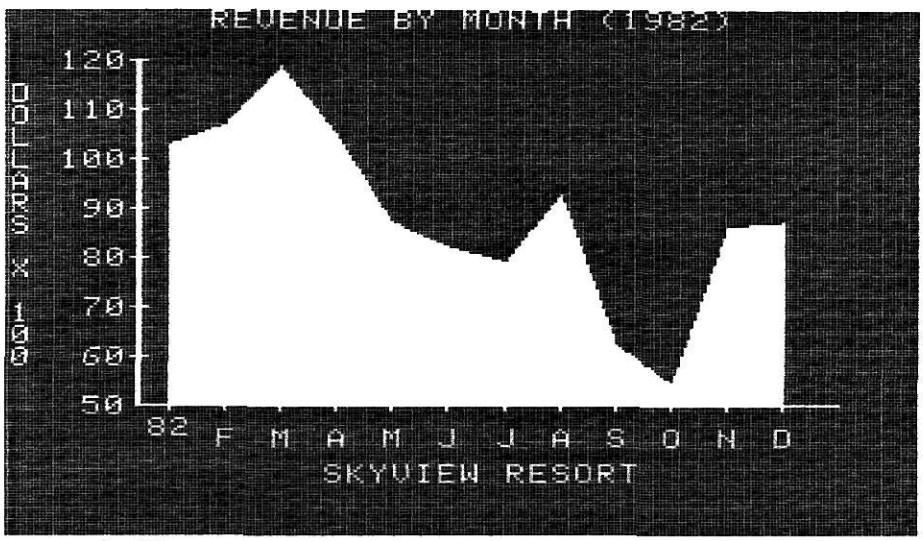
Once your file has been loaded, graphics of the type shown in Figure B.9 and Figure B.10 can be displayed using the VISIPLOT/VISITREND package. Note that these graphs were created using input from the DIF file shown in Figure B.8.

This ends PHASE IV.



LINE AND BAR CHARTS PRODUCED BY THE VISIPLOT/VISITREND GRAPHICS PACKAGE

FIG B.9



AREA CHART PRODUCED BY THE
VISIPLOT/VISITREND GRAPHICS PACKAGE

FIG B.10

DATA INTERCHANGE FORMAT FILES: A SIMPLE, DIRECT APPROACH TO PROVIDING TRANSPORTABLE GRAPHICS DATA

by

JAMES J. SHEEHY JR.

B.S., Ohio State University, 1975

AN ABSTRACT OF A MASTER'S REPORT

submitted in partial fullfillment of the

requirements for the degree

MASTER OF SCIENCE

Department of Computer Science

KANSAS STATE UNIVERSITY

Manhattan, Kansas

1983

ABSTRACT

This report describes a method by which Data Interchange Format (DIF) files are created on a host (Perkin-Elmer 8/32) computer and then transferred to an Apple II computer. Once transferred, the DIF file is used as a data input stream for the Visiplot/Visitrend graphics plotting program. The purpose of this project is to demonstrate the manner in which data-format standards can enhance the exchange of data between applications programs which use a common language and to emphasise how data-format standards can facilitate the inter-machine and the inter-language transfer of data. The DIF_BUILDER SYSTEM is utilized as a tool to demonstrate how DIF files accomplish these aims.