$1.50

Software Arts Technical Note
SATN-18

PROGRAMMER'S GUIDE
to
DIF™
A Data Interchange Format

Software Arts, Inc.
4/81 V1.5

Limited License to Copy

This Guide is intended for the use of the original purchaser only. The original purchaser is hereby licensed to copy it for his own use, provided that this notice is reproduced on each such copy. Copying of this Guide in any form for purposes of resale, license or distribution is prohibited.

No Warranty

This Guide is being published to enhance the usefulness of DIF, the data interchange format used by the VisiCalc® program and other programs. NEITHER SOFTWARE ARTS, INC. NOR PERSONAL SOFTWARE INC. MAKES ANY WARRANTY, EXPRESS OR IMPLIED, WITH RESPECT TO THE QUALITY, ACCURACY OR FREEDOM FROM ERRORS OF THE DIF FORMAT OR OTHER CONTENTS OF THIS DOCUMENT, INCLUDING, WITHOUT LIMITATION, ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR OF FITNESS FOR A PARTICULAR PURPOSE.

VisiCalc® a Trademark

The term "VisiCalc" is a trademark of Personal Software Inc. which designates a software product published by Personal Software Inc. under an exclusive license from Software Arts, Inc.

1. INTRODUCTION

It is often desirable to process the same data by more than one program. For example, a data management system may be used to record sales values. These values are then to be used as the basis for projections using the VisiCalc program. Finally the projections may be plotted by a third program. How can you get data from one program to another without requiring the user to type the data in anew each time? Each of the programs processing the data may be written by a different person, and may even run on different machines.

In order to allow programs to "talk" to each other, we must agree upon a standard language. Software Arts, Inc., the creators of the VisiCalc program, have developed a data interchange format, DIF, that can be used as a common language for data. This is the format in which VisiCalc saves data with the */S#* commands.

We are writing this document in order to explain to programmers how they can read and write data files using this format. The more programs that support the format, the more useful it becomes. The casual user should not be concerned about the details. It is only important to be aware that the format exists and that if two programs support the format, then it is likely that data produced by one can be processed by the other.

If you read this document fully, you will learn all of the details of the standard. This is not a tutorial, so you may find it helpful to skim the more technical parts that follow, and concentrate on the next section, the beginning of the Data File Format section, and the sample programs.

The sample programs in this document are all coded in a general dialect of BASIC, except as noted. Files are opened with an OPEN statement, and read and written with INPUT# and PRINT# statements. To get these programs to run on your system, you may have to modify them.

2. WHAT DIF DOES AND DOES NOT DO

The basic goal of DIF is to allow the interchange of data among a wide variety of programs. The type of data addressed by DIF is data that is stored in tables—columns and rows. Examples of this type of data would be time series, such as the daily closing price of one or more stocks that are to be input to a regression analysis package, or the actual expense figures for a company that are to be used as the starting point for a forecast. DIF treats all data as a group of equal length *vectors*—that is, groups of related data, like time series, or columns in a relation. The word vector is used, rather than column, since the actual orientation of the data (a horizontal row or vertical column) does not necessarily correspond to how it is logically oriented. Likewise, the corresponding elements of the vectors are called *tuples* rather than rows. For example, in the data below, the Sales, Cost and Profit figures (across the rows) could be viewed as vectors, with each year (down the columns) corresponding to a tuple:

```
Year        1980   1981   1982   1983
----        -----------------------
Sales        100    110    121    133
Cost          80     88     97    106
Profit        20     22     24     27
```

The actual choice of which grouping of the data is considered to be the vectors, and which the tuples, is really up to the programmer or user. Some programs may just view the data as a rectangle of unrelated data, while others may require the user to be aware of the grouping. The VisiCalc program would be an example of the former, and a plotting package would be an example of the latter.

In DIF, data is stored by tuples. That is, it consists of successive values from each vector grouped together into tuples, which are then output (or input) in that order. In the data used for our example, if the vectors were across the rows (Sales would be one vector, Cost and Profit the other two), then the first tuple would consist of the three numbers 100, 80, and 20, in that order. The second tuple would be 110, 88, and 22, and so on.

When the VisiCalc program deals with data in DIF it gives you the option of storing or loading "by rows" (R or RETURN) or "by columns" (C). What the VisiCalc program means by "by rows" is that the vectors go across the rows, and the tuples go down the columns. For example, in our example data, saving Sales, Cost and Profit by rows would output first the tuple 100, 80, 20, and then the tuple 110, 88, 22, etc. "By columns" is just the opposite, with the vectors down the columns, and the tuples across the rows. For the same data, the first tuple by columns would be 100, 110, 121, 133, and then 80, 88, 97, 106, etc.

Not all of the programs that process the data stored in DIF will have identical requirements. For example, some programs will only be able to process a simple list of numbers while others will want to store attributes associated with multiple vectors of numbers. Thus, a goal in the design of DIF was that programs should be able to keep descriptive information about the data, but must not be *required* to generate it. At the same time, the program reading the data should be able to ignore all descriptive information that is not relevant to the actual processing of data.

The primary constraint on the format of data stored in DIF is simplicity. It should be very simple for users to write programs in a common language to read and write data files. Since BASIC is so pervasive and minimal, the needs of BASIC were used to determine the details of the format. It is necessary for other languages, such as Pascal or PL/I, to be able to process this data, too. Fortunately these languages allow the use of subroutine libraries. Thus, a standard set of subroutines to process the interchange format can be provided for the users of those languages, freeing them from many of the details of processing the data.

Nongoals were just as important as goals during the design of DIF. Specifically, there is no emphasis on a minimal space representation. This representation is meant to be modest and does not attempt to preserve the richness available in many database systems. The central idea is that we should be able to transport a table of values (numeric and/or string) from one program to another. There is an additional mechanism to allow cooperating programs to exchange some information about the data, such as labelling.

Some of the more specific constraints are:

Predetermined data types

It is much simpler to write a program in BASIC if one knows ahead of time what the format of the data is, and in particular whether one is going to be reading a string or a number. Some BASICs are missing the VAL function that will convert from a string to a number, making it even more difficult. Therefore, DIF defines exactly which type of data is to be read at each point.

Lack of line input

Many BASICs do not have the ability to read a line of text without giving special meaning to some characters. For this reason strings containing special characters must be quoted.

Lack of parsing

Some BASICs will only input a whole line as a string. They do not use "," as a string value delimiter. Therefore, DIF always stores string values alone on a single line.

Input size

Many BASICs have a limited input buffer. 255 characters is a typical limit for the length of an input line. Therefore, DIF tries to keep most lines of information short.

Preallocation

In systems that permit dynamic allocation, it is often necessary to allocate the space before actually reading the data. Even when this is not required, knowing the total amount of data beforehand can be an important efficiency consideration. For this reason, DIF has a method for making this information available to a program reading the data.

End of data

In some systems it is either difficult or impossible to detect the end of data in a file gracefully. Thus the program should know when it has read the last value. DIF has a special provision to signal when the last data element has been read.

3. THE DATA FILE FORMAT

A DIF file consists of two parts—the header and the data part. The header describes the data and the data part has the actual values. An example of a DIF file is the following, which is from our sample data above. It has the vectors going across the rows, so there are three vectors, and four tuples. The various parts of the file are labelled, and will be described below:

```
TABLE
0,1
""
VECTORS     Header
0,3         Item
""                        Header
TUPLES
0,4
""
DATA
0,0
""
-1,0
BOT
0,100       1980 Sales
V
0,80        1980 Cost
V
0,20        1980 Profit
V
-1,0
BOT
0,110       Data
V           Value
0,88                      Tuple
V
0,22
V                                    Data
-1,0                                 Part
BOT
0,121
V
0,97
V
0,24
V
-1,0
BOT
0,133       1983 Sales
V
0,106       1983 Cost
V
0,27        1983 Profit
V
-1,0
EOD
```

THE HEADER

The header is organized into *header items.* Each header item contains a different piece of information about the data stored in the file. That data is sometimes numeric, and sometimes a string value.

STRUCTURE OF A HEADER ITEM

Each header item consists of four fields arranged as follows:

Topic
Vector number, Value
"String value"

The Topic

This is a keyword that identifies the header item. It must be a simple token readable as a string in BASIC without quotation marks. A word consisting of just letters with no spaces is best.

The Vector number

Several header items, such as a label, will apply to a specified vector. The Vector number specifies which vector this particular header item refers to. If the header item is not specific to a vector, such as a report title, this value should be 0.

The Value

This appears on the same line as the Vector number. It is used for header items that specify values, such as the number of vectors. It is zero if the value is not used by the header item. The value must be an integer.

The "String value"

This appears on a separate line after the Vector number and Value. It is used for header items that need string values rather than numeric values. The vector labels are an example. The string is always enclosed in quotes.

Thus the header item consists of three lines. The first line is the topic of the header item, the second line consists of two numbers and the third line has a string. The specific header items are described below.

Programs can ignore all header items until one with the topic DATA (described below) is found. The following program segment will skip the header items:

```
1000 INPUT#1,T$     :REM - Read the Topic name
1010 INPUT#1,S,N    :REM - Read the Vector #, Value
1020 INPUT#1,S$     :REM - Read the String value
1030 IF T$<>"DATA" THEN 1000 :REM - Check for
                             :REM - DATA header item
```

THE HEADER ITEMS

The standard header items are shown below with a description. The only required header items are TABLE and DATA, which must be the first and last header items, respectively.

TABLE
0,version
"title"

This is the first entry in the file. While it is not strictly required, it is important to allow for changes in future versions and it allows programs to verify that the file is a TABLE of data. The version number must be 1. Some programs may not accept the file without the TABLE header item.

VECTORS
0,count
" "

This tells how many data vectors are present. Some programs will require this header item to be present. If this header item is absent, the input program can calculate this value by counting the number of Data Values in each tuple (see below). N.B.: This header item must appear before header items that reference vector numbers, such as the LABEL header item.

TUPLES
0,count
" "

Specifies the length of each vector. (All vectors must be the same length.) Some programs will require this header item. If this header item is absent, the input program can calculate this value by counting the number of tuples before an end of data (EOD) Special Data Value (see below).

LABEL
vector#,line#
"label"

Provides a label for the specified vector. This is optional. The line# allows for labels spanning multiple lines, but can be ignored by systems allowing only single line labels. The values 0 and 1 should be equivalent for line#.

COMMENT
vector#,line#
"label"

This is similar to the LABEL header item for systems that allow an expanded description in addition to labels.

SIZE
vector#,#bytes
" "

This is used by programs, such as data base systems, that allocate fixed size fields for each value. Such programs, though, should be able to read files that do not contain SIZE information, since other programs may not be able to generate information of this type.

DATA
0,0
" "

This says that data follows. The data is organized by tuples, with one value from each vector in a given tuple.

Subsystems may define their own header items to meet their needs. Header items that will tend to be common should be standardized, such as the LABEL for a vector. The DIF Clearinghouse will serve as a repository for standard header items (see the address for DIF correspondence in the section Clearinghouse, below).

THE DATA PART

The data part consists of tuples, i.e. one value for each vector, in vector order. The tuples are made up of groups of two numeric values and one string value called *Data Values.* Each Data Value is used to represent the value of one element of data in the file.

In addition to the Data Values used to represent the actual data in the file, there are two types of Special Data Values used to provide information about the organization of the data. One Special Data Value is used to show where each tuple starts, and the other Special Data Value is used to indicate the end of all of the data in the file.

Data Values are all in the following format:

Type Indicator, Number Value
String Value

The first two fields are numeric values on a single line, the last is a string on a line by itself. These fields are:

The Type Indicator field

The Type Indicator is an integer that is used to indicate the way in which to interpret the rest of the fields in a Data Value. The currently assigned values for the Type Indicator are:

-1 Indicates that this Data Value is a Special Data Value, either a beginning of tuple indicator or an end of data indicator. See below for a discussion of the Special Data Values.

0 The data is numeric. The value of the Data Value is stored in the Number Value field, possibly modified by the String Value (see the descriptions of the Number Value and String Value fields below).

1 The data is a string. The value of the Data Value is stored in the String Value field.

2 This is an application specific value. The meaning is determined by the cooperating programs that are expected to use the data. For example, it might be an expression in the host language. For simple applications these values can be treated as strings.

The Number Value field

This is used when the Type Indicator is 0 to represent the value. The value must be a decimal (base 10) number. It may optionally be preceded by a sign (+ or –), have a decimal point, and immediately be followed by the letter E and an optionally signed power of ten exponent. The number may be preceded or followed by one or more blanks. Note that this is the only place in DIF where a non-integer value is allowed. Some programs that read data in DIF may only accept integer values (e.g., programs written in some BASICs or some systems programming languages).

The String Value field

The interpretation of this field depends upon the Type Indicator.

For normal Type Indicator 0 (numeric) data, the String Value should be the letter V (for value). If it is not V, then it is a Value Indicator, used to override the value. A subsystem may choose its own Value Indicators for named values, though they should be registered with the DIF Clearinghouse. The following Value Indicators are used by the VisiCalc program:

V

This is the normal case for numbers.

NA

This is a value marked explicitly as Not Available. The Number Value is set to 0.

ERROR

This is a value that represents the result of an invalid calculation, such as division by 0. The Number Value is set to 0.

It should always be possible to ignore the String Value for numeric data and just use the Number Value given. Another simple approach is to treat all values with a Value Indicator other than "V" as missing. Note that quotes are not permitted around the Value Indicator (for the sake of some BASICs).

For the Type Indicator of 1 (string data), this field is used for the string value itself. The quotes are optional if the field consists of just letters and does not contain any spaces. However, if a starting quote is given, a terminating quote must also be given.

Each tuple begins with a Special Data Value whose Type Indicator is -1, Number Value is 0, and whose String Value is BOT (for Beginning Of Tuple). This Special Data Value can be used by programs to determine how many vectors are in the file in the absence of a VECTORS header item (by counting the number of Data Values between BOT Special Data Values), or for a program to verify its position in a file.

At the end of the last tuple is a Special Data Value with a Type Indicator of -1, a Number Value of 0, and a String Value of EOD (for End Of Data). This will allow programs to determine the number of tuples in the absence of a TUPLES header item (by counting the number of tuples before an EOD Special Data Value), and to gracefully detect the end of the file.

4. SAMPLE PROGRAMS

Here are two sample programs. The first program creates a DIF file. The second program can read a DIF file and list its contents. They should be helpful in understanding how to manipulate DIF files. They are written as main programs with subroutines, so you can pick up code from them to be used in other programs. Both programs are written in a general BASIC, as described above.

CREATING A DIF FILE

```
100  REM - This program creates a  DIF  file.
110  REM - It prompts for the file name, number of vectors and
120  REM - tuples, and then for the values themselves.  Data
130  REM - may be either numeric (type 0) or string (type 1).
140  REM
1000 PRINT "FILE NAME";            :REM - Get name of file
1010 INPUT F$
1020 OPEN 1,F$                     :REM - Open for write
1030 PRINT "NUMBER OF VECTORS";    :REM - Get number of vectors
1040 INPUT NV                      :REM - into variable NV
1050 PRINT "NUMBER OF TUPLES";     :REM - and number of tuples
1060 INPUT NT                      :REM - into variable NT
1070 GOSUB 3000                    :REM - Write out DIF header
1080 FOR I = 1 TO NT               :REM - Get data and output it
1090   T = -1: V = 0: S$ = "BOT"   :REM - Output beginning of tuple
1100   GOSUB 4000
1110   FOR J = 1 TO NV             :REM - Get each Data Value
1120     PRINT "DATA TYPE FOR VECTOR #";J;", TUPLE #";I;
1130     INPUT T
1140     V = 0: S$ = "V"           :REM - Init values
1150     PRINT "DATA VALUE FOR VECTOR #";J;", TUPLE #";I;
1160     IF T=0 THEN INPUT V
1170     IF T=1 THEN INPUT S$
1180     GOSUB 4000                :REM - Output the Data Value
1190     NEXT J
1200   NEXT I
1210 T = -1: V = 0: S$ = "EOD"     :REM - Output end of data
1220 GOSUB 4000
1230 CLOSE 1
1240 PRINT "FINISHED CREATING DIF FILE ";F$
1250 STOP
3000                     :REM - Routine to write out DIF header
3010 PRINT#1,"TABLE": PRINT#1,"0,1": GOSUB 3500
3020 PRINT#1,"TUPLES": PRINT#1,"0,";NT: GOSUB 3500
3030 PRINT#1,"VECTORS": PRINT#1,"0,";NV: GOSUB 3500
3040 PRINT#1,"DATA": PRINT#1,"0,0": GOSUB 3500
3050 RETURN
3500                     :REM - Routine to write "" (null string)
3510 PRINT#1,CHR$(34);CHR$(34)     :REM - See Appendix on quoted
3520 RETURN                        :REM - strings in BASIC, below
4000                     :REM - Routine to write out Data Value
4010 PRINT#1,T;",";V
4020 PRINT#1,S$
4030 RETURN
4040 END
```

Note that if the string values being saved have spaces or special characters, the code at line 4020 should be changed to check for those cases, and add leading and trailing quotes. See the discussion about Quoted Strings in BASIC in the Appendix.

LISTING A DIF FILE

```
100  REM - This program reads a DIF file
110  REM - and lists its contents.  The program prompts for
120  REM - the name of the file to be listed.
500  DIM T(100)                    :REM - Maximum of 100 vectors
510  DIM V(100)                    :REM - T, V, and V$ hold the
520  DIM V$(100)                   :REM - Type Indicator, Number
530                                :REM - Value and String Value
540                                :REM - of each element in a tuple
550                                :REM -
1000 GOSUB 5000                    :REM - Call initialization code
1010 GOSUB 6000                    :REM - Read header
1020 FOR I = 1 TO NT               :REM - Read all of the tuples
1030   PRINT "VALUES FOR TUPLE #";I
1040   GOSUB 7000                  :REM - Get a tuple
1050   FOR J = 1 TO NV             :REM - Output each element
1060     IF T(J)=0 THEN PRINT V(J) :REM - Output numeric value
1070     IF T(J)=1 THEN PRINT V$(J):REM - Output string value
1080     NEXT J
1090   NEXT I
1100 CLOSE 2
1110 PRINT "FINISHED LISTING FILE ";F$
1120 STOP
5000                     :REM - Initialization code
5010 PRINT "FILE NAME";            :REM - Get name of file to read
5020 INPUT F$
5030 OPEN 2,F$                     :REM - Open file for read
5040 NV = 0                        :REM - Init counts of vectors
5050 NT = 0                        :REM - and tuples
5060 RETURN
6000                     :REM - Read header, and set NV and NT
6010 INPUT#2,T$                    :REM - Get Topic name
6020 INPUT#2,S,N                   :REM - Get Vector number, Value
6030 INPUT#2,S$                    :REM - Get "String value"
6040 IF T$="VECTORS" THEN 6500     :REM - Check for known header
6050 IF T$="TUPLES" THEN 6600      :REM - items
6060 IF T$="DATA" THEN RETURN      :REM - DATA ends header
6070 GOTO 6010                     :REM - Ignore unknown ones
6500 NV = N                        :REM - Value is number of vectors
6510 PRINT "THE FILE HAS ";NV;" VECTORS."
6520 IF NV<=100 THEN 6010          :REM - If not too many, continue
6530 PRINT "TOO MANY VECTORS.  THIS PROGRAM ONLY HANDLES 100."
6530 CLOSE 2
6540 STOP
6600 NT = N                        :REM - Value is number of tuples
6610 PRINT "THE FILE HAS ";NT;" TUPLES."
6620 GOTO 6010                     :REM - Get next header item
7000                     :REM -  Get all vector elements in a tuple
7010 GOSUB 8000                    :REM - Get next Data Value
7020 IF T1<>-1 THEN 9000           :REM - Must be BOT or else error
7030 IF S$<>"BOT" THEN 9000
7040 FOR K = 1 TO NV               :REM - Get each Data Value
7050   GOSUB 8000
7060   IF T1=-1 THEN 9000
7070   V(K) = V1                   :REM - Save Values and Type
7080   V$(K) = S$                  :REM - Indicator
7090   T(K) = T1
7100   NEXT K
7110 RETURN
8000                     :REM - Get next Data Value
8010 INPUT#2,T1,V1                 :REM - Get Type Indicator,
8020 INPUT#2,S$                    :REM - Numeric Value and String
8030 RETURN                        :REM - Value
```

```
9000 PRINT "ERROR IN FILE FORMAT."
9010 CLOSE 2
9020 STOP
9030 END
```

Please note that while the above program can read many DIF files correctly, it depends upon the TUPLES and VECTORS header items to determine the organization of the file. A more general program could be written that, in the absence of these header items, deduced their values from the placement of BOT and EOD Special Data Values. While most programs that deal with DIF should be able to produce TUPLES and VECTORS header items (the VisiCalc program, for example, does), some may not (such as a program that records data incrementally, and doesn't know how many data points it will encounter until it is finished).

5. APPENDICES

QUOTED STRINGS IN BASIC

Writing the quoted strings is not always convenient in BASIC. In some implementations, quotes may be included in a string by doubling them. For example:

```
PRINT#1,"TABLE"
PRINT#1,0,1
PRINT#1,"""Stock Prices for ABC Computer Co."""
```

In other implementations the CHR$ function must be used:

```
PRINT#1,"TABLE"
PRINT#1,0,1
PRINT#1,CHR$(34);"Stock Prices for ABC Computer Co.";CHR$(34)
```

Apple Integer BASIC presents special problems. It seems that it is necessary to POKE an assembly language routine into memory to output a quote. The following sequence will setup such a program at location $300 (hex):

```
100 POKE 768,169:POKE 769,162           :REM - LDA #´"´+$80
110 POKE 770,108:POKE 771,54:POKE 772,0 :REM - JMP (CWSL)
```

And to use this code:

```
120 PRINT "TABLE"
130 PRINT 1,0
140 CALL 768
150 PRINT "Stock Prices for ABC Computer Co.";
160 CALL 768
170 PRINT
```

Apple Integer BASIC also requires that the user remove the quotes from the input string with:

```
300 IF LEN(S$) > 2 THEN
         IF (ASC(S$(1,1)) MOD 128) = 34 THEN
              S$ = S$(2,LEN(S$)-1)
```

This assumes that there is also a trailing quote. Note that in order to make the quoted string itself acceptable to most BASICs it must not contain a quote.

CHARACTER SETS

The character set is assumed to be that of the host machine. Thus, if one is transferring a file from a machine using ASCII to one using EBCDIC, the appropriate conversions must be made. In addition, some machines may require that the quote be changed to an apostrophe. These changes should be transparent to most users. In order to assure compatability, strings should not contain nonprinting characters, other than the end of line sequence (RETURN, CR/LF, NEWLINE or whatever).

The ASCII character set defines 95 printable characters. The user should be aware that some systems do not make it easy to use the full set. In particular, keywords (including topic names and number types) must be in upper case. Some systems only support a limited set of characters, often 64 printable characters or less. When transporting a file to such a system the upper and lower case characters would be mapped together to one case. Other special characters may be mapped into common characters. If these transformations affect the integrity of the data, it should be specified in the documentation associated with the data.

6. CLEARINGHOUSE

In order to coordinate information about DIF and the programs that make use of it, Software Arts, Inc. is setting up a clearinghouse for such information.

We would appreciate it if the authors of programs that support DIF would send a one page description of the program to the clearinghouse. This description should include a short write-up of what the program does, on which computers it runs, how it relates to DIF, and how it may be obtained.

Users who would like a copy of the information that we receive should send $6.00 (to cover the costs of running the clearinghouse and providing the information) and their name, address and zip code, with a note specifically requesting a copy of the list of programs that support DIF.

All correspondence relating to DIF should be sent to the following address:

DIF Clearinghouse
P.O. Box 527
Cambridge, MA 02139