GENERAL DESCRIPTION OF THE PREP SUBPROGRAM

The PREP subprogram is a subroutine resident in the Personal Record Keeping and Statistics command modules. When the command module is plugged in, this routine can be called from a BASIC program. It is used to define a fixed length data area in the VDP RAM. This area can then be used by the other subprograms to manipulate data files. The CALL statement is used to execute the routine and takes the following form:

CALL P(V)

where V = number of bytes to reserve {numeric expression}

V is a numeric constant, variable, or expression which specifies the number of bytes to be reserved for the data area. The maximum size for a Personal Record Keeping file is 12048 bytes. However, the disk controller reserves some VDP RAM for disk I/O. When a disk controller is attached to the system, the maximum size for a PRK file drops to 10000 bytes. For a Statistics file the maximums are 7440 bytes without the disk controller and 5392 bytes with the disk controller. When using the PREP subprogram to allocate a data area, the maximum values are as follows:

With no disk controller or

After CALL FILES(0).....13820

After CALL FILES(1).....12768

After CALL FILES(2).....12250

For each additional file subtract 518 bytes.

If V evaluates to a value greater than the appropriate maximum, a "NOT ENOUGH MEMORY" error will occur.

In order to avoid problems with variables that may have already been allocated, a call to PREP should always be done imperatively, and should be followed by a NEW command to avoid unpredictable results. If PREP is called with a BASIC program residing in VDP RAM, an "ILLEGAL CALL" error will occur.

Once a data area has been allocated with the PREP subprogram, that area will remain allocated until BASIC is exited through a BYE command or by pressing FCTN +.

GENERAL DESCRIPTION OF THE LOAD SUBPROGRAM

The LOAD subprogram is a subprogram resident in the Personal Record Keeping and Statistics command modules. When the command module is plugged in, this routine can be called from a BASIC program. It is used to load a data file from an external device into the data area reserved by the PREP subprogram. The CALL statement is used to execute the routine and takes the following form:

```
CALL L(V$,V)
```

where

```
V$ = file-name          {string expression}

V = return variable     {numeric variable}
```

V$ is a string constant, variable, or expression which specifies the file to be loaded. V is a numeric variable in which the LOAD routine returns a code to indicate whether the LOAD was or was not successful. V=0 indicates that an error occured. Any other value indicates that the LOAD was successful. The following conditions will cause a LOAD failure.

1. The specified file and/or device does not exist.
2. No data area has been allocated.
3. The defined data area is too small for the data to be loaded.
4. General I/O errors.

CAUTION

Personal Record Keeping, Statistics, and the LOAD and SAVE routines use "program" type files. This means that 256 byte records are used. Thus the "effective" size of a file is the true size of the file rounded up to the nearest multiple of 256. For this reason it is possible to save a file which cannot be reloaded into the same size data area.

GENERAL DESCRIPTION OF THE SAVE SUBPROGRAM

The SAVE subprogram is a subprogram resident in the Personal Record Keeping and Statistics command modules. When the command module is plugged in, this routine can be called from a BASIC program. It is used to save a data file from an external device into the data area reserved by the PREP subprogram. The CALL statement is used to execute the routine and takes the following form:

CALL L(V$,V)

where

V$ = file-name {string expression}

V = return variable {numeric variable}

V$ is a string constant, variable, or expression which specifies the file in which to save data. V is a numeric variable in which the SAVE routine returns a code to indicate whether the SAVE was or was not successful. V=0 indicates that an error occured. Any other value indicates that the save was successful. The following conditions will cause a SAVE failure.

1. The specified file and/or device does not exist.

2. No data area has been allocated.

3. General I/O errors.

CAUTION

Personal Record Keeping, Statistics, and the LOAD and SAVE routines use "program" type files. This means that 256 byte records used. Thus the "effective" size of a file is the true size of the file rounded up to the nearest multiple of 256. For this reason it is possible to save a file which cannot be reloaded into the same size data area.

GENERAL DESCRIPTION OF THE ACCEPT SUBPROGRAM

The ACCEPT subprogram is a subroutine resident in the Personal Record Keeping and Statistics command modules. When the command module is plugged in, this routine can be called from a BASIC program. It is used to accept data entry from the console keyboard, and echo that entry on the screen at a specified location. The CALL statement is used to execute the routine and takes one of the following forms:

1. CALL A(Y,X,W,C,V,L,H)

2. CALL A(Y,X,W,C,V)

3. CALL A(Y,X,W,C,V,F)

4. CALL A(Y,X,W,C,V$)

where

Y =	Y-screen position	(numeric expression)
X =	X-screen position	(numeric expression)
W =	field width	(numeric expression)
C =	return code	(numeric variable)
V =	return variable	(numeric variable)
V$=	return variable	(string variable)
L =	low value	(numeric expression)
H =	high value	(numeric expression)
F =	field number	(numeric expression)

Y specifies the vertical position on the screen. The range is from 1 for the top line of the screen to 24 for the bottom line of the screen. X specifies the horizontal position on the screen. The range is from 1 for the left most position to 28 for the right most position. If the value of either of these parameters exceeds the maximum position, then the position is calculated minus the maximum position. Thus, if Y = 34 and X = 60, the data will be displayed at Y = 10 and X = 4. If either position parameter is less that 1, it is set to 1. Thus, Y = 0 and X = -5 would give the position Y = 1 and X = 1.

W specifies the field width. If its value is greater than the remaining number of spaces on the specified line, the field is truncated at the right edge of the screen.

C is a numeric variable in which a code is returned. The code returned is an integer from 1 to 7 as follows:

1 = valid non-empty data was entered

2 = an empty field was entered

3 = FCTN 7 was pressed

4 = FCTN 8 was pressed

5 = FCTN 6 was pressed

6 = FCTN 5 was pressed

7 = FCTN 9 was pressed

This return code makes it possible to process "missing" data (i.e. null entries) and the five special function keys listed. The most common method of handling these is to follow the CALL A( ) statement with an ON-GOTO statement, for example:

```
ON C GOTO 580,620,920,720,680,800,100
```

Thus control can be passed to the proper location depending on the action taken in the ACCEPT routine. If only one or two of the 7 listed actions are allowed from a given CALL, the transfer of control may be better handled by one or two IF-THEN statements in place of the ON-GOTO statement, for example:

```
IF C = 5 THEN 680
IF C > 1 THEN 420
```

V or V$ is the variable in which the data entered is returned.

L and H are optional low-high value parameters. They establish a range which the ACCEPT routine will use to check the validity of the data entered. If an attempt is made to enter a value outside the specified range, an error beep is given, and the cursor returns to the beginning of the field. If these two optional parameters are not specified, no range check will be performed.

The parameter F is only valid when there exists a completely defined Personal Record Keeping of Statistics file header. In this event, F specifies the field number of the data being entered. The ACCEPT routine will retrieve the field characteristics (type, width, number of decimal places, etc.), and verify that the data entered is valid.

For example, a decimal point will not be allowed if the field is integer type; scientific notation will not be allowed if the field is integer or decimal type; or 3 decimal places will not be allowed if 2 decimal places are specified.

WARNING

If a valid Personal Record Keeping or Statistics file header does not exist, the results of using this field parameter are totally unpredictable.

GENERAL DESCRIPTION OF THE DISPLAY SUBPROGRAM

The DISPLAY subprogram is a subroutine resident in the Personal Recording Keeping and Statistics command modules. When the command module is plugged in, this routine can be called from a BASIC program. It is used to display numeric values, character strings, or special characters (if previously defined) on the screen at a specified location. The CALL statement is used to execute the routine and takes one of the following forms:

1. CALL D(Y,X,W,V)

2. CALL D(Y,X,W,V$)

3. CALL D(Y1,X1,W1,V1,Y2,X2,W2,V2$,Y3,X3,W3,V3,etc.)

where

Y = Y-screen position	{numeric expression}
X = X-screen position	{numeric expression}
W = field width	{numeric expression}
V = display value	{numeric expression}
V$= display value	{string expression}

Y specifies the vertical position on the screen. The range is from 1 for the top line of the screen to 24 for the bottom line of the screen. X specifies the horizontal position on the screen. The range is from 1 for the left most position to 28 for the right most position. If the value of either of these parameters exceeds the maximum position, then the position is calculated minus the maximum value. Thus, if Y = 34 and X = 60, the data will be displayed at Y = 10 and X = 4. If either position parameter is less than 1, it is set to 1. Thus, Y = 0 and X = -5 would give the position Y = 1 and X = 1.

W specifies the field width. If its value is positive, the field is cleared before the new value is displayed. If W is negative, the absolute value is used as the field width, but the field is not cleared. If the specified field width is less than the width of the data to be displayed, the data is truncated. If the specified field width is greater than the remaining number of spaces on the specified line, the field is truncated at the right edge of the screen.

V is any type of numeric expression. The expression will be evaluated, and the result will be displayed. Similarly, V$ is any type string expression, it will be evaluated and the resulting string displayed.

*Note that multiple displays may be done with one DISPLAY call (form 3 above). The position, field width, and display values are simply listed in the sequence desired. The only limit here is the length of the line in BASIC.

GENERAL DESCRIPTION OF THE GETPUT SUBPROGRAM

The GETPUT subprogram is a subroutine resident in the Personal Record Keeping and Statistics command modules. When the command module is plugged in, this routine can be called from a BASIC program. It is used to write data to and read data from a file which has been defined using the PREP and HEADER subprograms. The CALL statement is used to execute this routine and takes one of the following forms:

1. CALL G(R/W, REC, FLD, V)

2. CALL G(R/W, REC, FLD, V$)

3. CALL G(R/W, REC, FLD, MIS, V2)

4. CALL G(R/W, REC, FLD, MIS, V2$)

where

R/W	= read/write code	{numeric expression}
REC	= record number	{numeric expression}
FLD	= field number	{numeric expression}
V	= numeric value	{numeric expression}
V$	= string value	{string expression}
MIS	= return code	{numeric expression}
V2	= return variable	{numeric variable}
V2$	= return variable	{string variable}

The value of R/W determines whether a read or write is done. The values are:

0 = write valid data to the file

1 = read data from the file

2 = indicate missing data in file

Any other value for R/W will cause an error.

REC is the record number. In Personal Record Keeping this is referred to as page number, and in Statistics as observation number.

If the value is higher than the highest possible record number, an error occurs. If a write is done to a higher record than the highest previously defined record, the higher record number is stored in the header as the new highest record number. If a read is done from an undefined record, the results are unpredictable. Therefore, it is important to create records sequentially without skipping any. A negative number or zero also yields unpredictable results.

FLD is the field number within the specified record. In Personal Record Keeping this is referred to as item number, and in Statistics as variable number. Each record has the same defined fields. If the field number is higher than the highest defined field number an error will occur. A negative or zero value will generate unpredictable results.

V and V$ are values to be written to the field. If they are expressions, the result of evaluating the expression is stored. If the value is incompatible with the definition of the specified field, the results are unpredictable. For example, if a field is defined to be decimal type with a width of 6 and 2 decimal places, both 7654 and 7.654 are incompatible values. (7654 has 1 too many digits to the left of the decimal point, and 7.654 has 1 too many digits to the right of the decimal point.) When indicating missing data in the file, V or V$ must be included in the CALL parameters. However, the value of V ar V$ is ignored.

MIS is a numeric variable in which a code is returned to indicate normal versus missing data. This parameter must be included when reading from the file, but must not be included when writing. After a read MIS = 0 indicates data found, MIS = 1 indicates missing data.

V2 is a numeric variable in which the numeric value being read from the file is returned. V2$ is a string variable in which a string value is returned. If a numeric variable is used when reading a string item, or a string variable is used when reading a numeric item, an error will occur. If missing data is found by the read, the return variable is not changed.

GENERAL DESCRIPTION OF HEADER SUBPROGRAM

The HEADER subprogram is a subroutine resident in the Personal Record Keeping and Statistcs command modules. When the command module is plugged in, this routine can be called from a BASIC program. It is used to write and/or read the information in the file header. The CALL statement is used to execute the routine and takes one of the following forms:

1. CALL H(R/W, INFO, FLD, V)

2. CALL H(R/W, INFO, FLD, V$)

where

R/W	= access code	{numeric expression}
INFO	= header item number	{numeric expression}
FLD	= field number	{numeric expression}
V	= variable name	{numeric expression}
V$	= variable name	{string expression}

R/W is the read/write code as follows:

0 = write information in header

1 = read information in header

INFO contains a value from 1 to 14 which specifies which item of header information is to be read/written. The item structure of the header is as follows:

1. File name
   0 to 9 characters

2. Day of month
   Integer from 1 to 31

3. Month
   Integer from 1 to 12

4. Year
   Integer from 0 to 99

5. Number of fields per record
   This item is automatically updated by the HEADER routine each time a new highest numbered field is defined.

6. Number of records
   This item is automatically updated by the GETPUT routine each time a new highest numbered record is written.

7. Length of header in bytes
   This item is automatically maintained by the HEADER routine.

8. Length of each record in bytes
   This item is automatically maintained by the HEADER routine.

9. Name of field
   0 to 9 charaters

10. Type of field

    1 = characters

    2 = integer

    3 = decimal

    4 = scientific notation

11. Width of field

    character : 1 to 15

    integer : 1 to 10

    decimal : 2 to 11

    scientific notation : 8 to 13 (width of scientific notation field is automatically handled by HEADER routine)

12. Number of decimal places for field

    character : 0 (handled by HEADER routine)

    integer : 0 (handled by HEADER routine)

    decimal : 1 to width -1

    scientific notation : 0 to 5

13. Amount of storage for field in bytes
    This item is automatically maintained by the HEADER routine.

14. Position of field in record
    This item is automatically maintained by the HEADER routine.

NOTE

Items 9 through 14 are repeated for each field defined.

FLD specifies the field number. This parameter is ignored for items 1 through 8 but must be included in the parameter list for the CALL statement.

When writing information to the header, V and/or V$ are constants, variables, or expressions containing the information to be stored in the header. When reading information from the header, V and/or V$ are the variables where the information is returned.

SOME SAMPLE BASIC PROGRAMS

The Personal Record Keeping and Statistics command modules do not store their data in the standard BASIC format for numerics and strings. The following BASIC program converts a Personal Record Keeping file into two standard BASIC files. The first is a HEADER file and the second is the DATA file. This program assumes that the files are on disk.

* Execute the following three BASIC commands imperatively.

```
> CALL FILES(1)

> CALL P(10000)

> NEW
```

* Run the following program. Each line is preceded by a comment about the purpose of the statement.

```
* Load the data file into the reserved area.

100 CALL L("DSK1.PRKFILE",C)

* Check error indicator.  0 = failure, non 0 = success.

110 IF C = 0 THEN 450

* Open HEADER file.

120 OPEN #1:"DSK1.PRKHEADER",RELATIVE,INTERNAL,OUTPUT,FIXED

* Read internal file name.

130 CALL H(1,5,0,F$)

* Read number of fields per record.

140 CALL H(1,5,0,F)

* Read number of records.

150 CALL H(1,6,0,R)

* Print information to Header file.

160 PRINT #1:F$,F,R

* Print information on screen.
```

```
170 PRINT F$,F,R
* Set up a loop to read field definitions.
180 FOR I=1 TO F
* Read field name.
190 CALL H(1,9,I,F$)
* Read field type.
200 CALL H(1,10,I,T)
* Read field width.
210 CALL H(1,11,I,W)
* Check type of item.
220 IF T<>1 THEN 250
*Field is alpha so add width + 1.
230 S = S + W + 1
*Skip around.
240 GOTO 260
* Field is numeric so add 9.
250 S = S + 9
* Read number of decimal places.
260 CALL H(1,12,I,D)
*Print information to Header file.
270 PRINT #1, REC I:F$,T,W,D
* Print information to screen.
280 PRINT F$;T;W;D
*End of loop.
290 NEXT I
```

```
*Close Header file.
300 CLOSE #1
*Open DATA file.
310 OPEN#1:"DSK1.PRKDATA", RELATIVE,INTERNAL,OUTPUT,FIXED S+2
*Set up loop to read records and print them to DATA file.
320 FOR I=1 TO R
*Print record number to screen.
330 PRINT I
*Set up loop to read fields for each record.
340 FOR J=1 TO F
*Read field type.
350 CALL H(1,10,J,T)
*Check field type.
360 IF T=1 THEN 430
*Field is numeric so read into numeric variable.
370 CALL G(1,I,J,C,D)
*Check missing data code.
380 IF C=0 THEN 400
*Missing numeric data so store negative infinity. (How missing data is
handled is a matter of personal choice.)
390 D=-9.999999999999E+127
*Data found so print it.
400 PRINT#1:D,
*Print data to screen.
410 PRINT D;
```

```
170 PRINT F$,F,R
* Set up a loop to read field definitions.
180 FOR I=1 TO F
* Read field name.
190 CALL H(1,9,I,F$)
* Read field type.
200 CALL H(1,10,I,T)
* Read field width.
210 CALL H(1,11,I,W)
* Check type of item.
220 IF T<>1 THEN 250
*Field is alpha so add width + 1.
230 S = S + W + 1
*Skip around.
240 GOTO 260
* Field is numeric so add 9.
250 S = S + 9
* Read number of decimal places.
260 CALL H(1,12,I,D)
*Print information to Header file.
270 PRINT #1, REC I:F$,T,W,D
* Print information to screen.
280 PRINT F$;T;W;D
*End of loop.
290 NEXT I
```

```
*Close Header file.
300 CLOSE #1
*Open DATA file.
310 OPEN#1:"DSK1.PRKDATA", RELATIVE,INTERNAL,OUTPUT,FIXED S+2
*Set up loop to read records and print them to DATA file.
320 FOR I=1 TO R
*Print record number to screen.
330 PRINT I
*Set up loop to read fields for each record.
340 FOR J=1 TO F
*Read field type.
350 CALL H(1,10,J,T)
*Check field type.
360 IF T=1 THEN 430
*Field is numeric so read into numeric variable.
370 CALL G(1,I,J,C,D)
*Check missing data code.
380 IF C=0 THEN 400
*Missing numeric data so store negative infinity. (How missing data is
handled is a matter of personal choice.)
390 D=-9.999999999999E+127
*Data found so print it.
400 PRINT#1:D,
*Print data to screen.
410 PRINT D;
```

```
*Skip around.

420 GOTO 400

* Field is alpha so read into string variable.

430 CALL G(1,I,J,C,F$)

* Check missing data code.

440 IF C = 0 THEN 460

* Missing string data so store a space character.  (How missing data is
  handled is a matter of personal choice.)

450 F$ = " "

* Data found so print it.

460 PRINT #1:F$,

* Print data to screen.

470 PRINT F$; " " ;

* End of field loop.

480 NEXT J

* Finish printing the record.

490 PRINT #1:" "

* Finish pending screen print.

500 PRINT

* End of record loop.

510 NEXT I

* Close the Data file.

520 CLOSE #1

* When the file is closed then stop.

530 STOP

* Indicate an error in trying to load PRK file.
```

```
540 PRINT "ERROR IN LOADING PRK FILE"

* Stop the program.

550 STOP
```

The results of running this program are:

1. A. HEADER file containing -

    a. the PRK file name, number of fields per PRK record, and number of PRK records in record 0,

    b. and the definition (name, type, width, decimal places) of field #n in record n.

2. A DATA file containing PRK record #n in record n-1.

The following BASIC program takes the HEADER and DATA files created in the previous example and converts them into a PRK file which can be saved by the SAVE routine.

* Execute the following three BASIC commands imperatively.

> CALL FILES(1)

> CALL P(10000)

> NEW

* Run the following program. Each line is preceded by a comment about the purpose of that statement.

```
* Open the HEADER file.
100 OPEN #1:"DSK1.PRKHEADER", RELATIVE, INTERNAL, INPUT, FIXED
* Read file name, number of fields per record, and number of records.
110 INPUT #1:F$,F,R
* Print information to screen.
120 PRINT F$,F,R
* Write file name to PRK header.
130 CALL H(0,1,0,F$)
* Read number of fields per record.
140 CALL H(0,5,0,F)
* Read number of records.
150 CALL H(0,6,0,R)
* Set up loop to create remainder of PRK header.
160 FOR I=1 TO F
* Read field name, type, width, and decimal places.
170 INPUT #1, REC I:F$,T,W,D
* Print information to screen.
180 PRINT F$;T;W;D
* Write field name to PRK header.
```

```
190 CALL H(0,9,I,F$)
* Write field type to PRK header.
200 CALL H(0,10,I,T)
* Check type.  For scientific notation do not write width.
210 IF T=4 THEN 240
* Write field width to PRK header.
220 CALL H(0,11,I,W)
* Write number of decimal places to PRK header.
230 CALL H(0,12,I,D)
* Check type.  For character and integer fields do not write decimal
  places.
240 IF T < 3 THEN 250
* End of loop.
250 NEXT I
* Close the Header file.
260 CLOSE #1
* Open the Data file.
270 OPEN #1:"DSK1.PRKDATA", RELATIVE, INTERNAL, INPUT, FIXED
* Set up loop to read data records and write them to PRK file.
280 FOR I=1 TO R
* Print record number to screen.
290 PRINT I
* Set up loop to read fields for each record.
300 FOR J=1 TO F
* Read field type.
310 CALL H(1,10,J,T)
* Check field type.
```

```
320 IF T=1 THEN 400
* Field is numeric so read into numeric variable.
330 INPUT #1:D,
* Print data to screen.
340 PRINT D;
* Check whether this should be represented as missing data.
350 IF D=-9.999999999999E+127 THEN 380
* Normal data so write it to PRK file.
360 CALL G(0,I,J,D)
* Continue loop.
370 GOTO 440
* Indicate missing numeric data.
380 CALL G(2,I,J,D)
* Continue loop.
390 GOTO 460
* Field is alpha so read into string variable.
400 INPUT #1:F$
* Print data to screen.
410 PRINT F$; " " ;
* Check for missing data.
420 IF F$=" " THEN 450
* Normal data so write it to PRK file.
430 CALL G(0,I,J,F$)
* Continue loop.
440 GOTO 460
* Indicate missing string data.
```

```
450 CALL G(2,I,J,F$)
* End of field loop.
460 NEXT J
* Finish the record.
470 INPUT #1:F$,D$
* Finish pending print to screen.
480 PRINT F$,D$
* End of record loop.
490 NEXT I
* Close the Data file.
500 CLOSE #1
* Save the PRK file.
510 CALL S("DSK1.PRKFILE",C)
* Check error indicator.  0 = failure, non-0 = success.
520 IF C THEN 540
* Indicate an error in trying to save PRK file.
530 PRINT "ERROR IN SAVING PRK FILE"
* Stop the program.
540 STOP
```