dBASE II

Assembly-Language
Relational Database Management
System

VOL. II ***REFRENCE MANUAL***

Scanned by cvxmelody
http://www.cvxmelody.net/AppleUsersGroupSydneyAppleIIDiskCollection.htm

With thanks to Danny C.

AJR

CONTENTS

1.0 USING dBASE

To execute the dBASE program, place the dBASE distribution diskette (or preferably, a copy of that diskette) into any available disk drive. Set that drive to be the default drive (e.g. if the disk is placed into the "B" drive, type in "B:" followed by a carriage return) and then type in the following line:

```
DBASE
```

The program will then be loaded into memory, and will start execution with a date request:

```
ENTER DATE AS MM/DD/YY  OR RETURN FOR NONE:
```

This date will be posted on any database that is altered during the following run and will also be printed in REPORT headings for any report generated in that run. The date is checked for calendar accuracy. WARNING: The calendar check is not valid for February 29 in the years 1900 and 2100. A slash or any special character (except a period) may be used to delimit the numbers.

Examples of valid dates:

```
1,1,81
02 02 82
3/17/83
```

Then the sign-on message is displayed:

```
***  dBASE II    VER 2.xxx***
```

The period on the second line is the dBASE prompt, indicating that dBASE is ready to accept commands. Commands to dBASE are generally imperative sentences: a verb possibly followed by phrases that give further direction about the action to be taken. dBASE scans each line completely before executing any part of it. If dBASE detects an error in the command then the user is notified via error messages on the console. Generally, the user may correct the erroneous command and re-issue rather than re-enter the entire command. When dBASE detects an error that it can't describe explicitly, it assumes that the error is a syntax error and displays the erroneous line with a question mark at the beginning of the phrase that caused the confusion.

Error recovery examples:

```
. DISPRAY MEMORY
*** UNKNOWN COMMAND
DISPRAY MEMORY                  erroneous command echoed
CORRECT AND RETRY? Y            Yes, correct
CHANGE FROM :PR                 change the letters PR
CHANGE TO   :PL                 to PL
DISPLAY MEMORY                  after the change
MORE CORRECTIONS? (cr)          return = no more changes

. STORE (2+2 TO X
*** SYNTAX ERROR ***
     ?                          the string (2+2 is indicated
STORE (2+2 TO X
CORRECT AND RETRY? Y
CHANGE FROM :+2
CHANGE TO   :+2)
STORE (2+2) TO X
MORE CORRECTIONS? N             N(o) more changes
  4                             the result

. SUM TO X
NO EXPRESSION TO SUM            explanation
SUM TO X
CORRECT AND RETRY? N            no change, abort this command
```

The program can also be executed in the following manner:

DBASE <filename>

This will load dBASE into memory, access a command file <filename>, and begin immediate execution of that command file. This form is especially useful when using dBASE in a SUBMIT file or when using the chaining option of the dBASE QUIT command.

CONTROL CHARACTERS

ctl-P	- Toggles print switch (see also SET PRINT command)
ctl-U	- Deletes current line
ctl-X	- Deletes current line (except in full screen edit)
Rubout	- Deletes last character entered
ctl-H (or backspace)	- Deletes the last character entered
ESC	- Escapes from certain possibly long-running commands. I.e. DISPLAY, COUNT, DELETE, INPUT, LIST, LOCATE, RECALL, REPLACE, SKIP, and SUM. Also ESC serves as an escape from ACCEPT, INPUT, REPORT (dialogue), and WAIT. In all cases, ESC returns control to the interactive monitor and displays a dot prompt.

When in a command file execution, dBASE checks for an ESC character before starting every command line.

NOTE: This escape capability can be disabled by the SET ESCAPE OFF command.

2.0 SYSTEM REQUIREMENTS

In order for dBASE to operate properly, a system with the following attributes should be made available.

a) 8080 or Z-80 based microprocessor system;

b) 48K bytes (or more) of memory including CP/M (dBASE uses memory up to A400 hex). Note: on some machines, including Apple, Heath, and Northstar, more than 48K is required because of an oversized CP/M module;

c) CP/M operating system (version 1.4 or 2..

d) One or more mass storage devices operating under CP/M (usually floppy or rigid disk drives);

e) A cursor addressable CRT device (preferably a 24 line by 80 column CRT) if full screen operations are to be used;

f) Optional text printer (for some commands).

3.0 dBASE FILES

Basically, a file is a collection of information residing on a mass storage device that contains the user's data. The information can be stored to or retrieved from the file. Files can be grouped into six types, each one either concerned with a particular operation of or created by dBASE.

All dBASE files are standard CP/M files with a name field of eight characters and a file type of three characters. Listed below are the default file types used by dBASE. For each command that accesses a file, the type field may be left off and dBASE will assume the default type for that command. For instance, if a database file already has DBF as its type, then it need not be specified in any of the file manipulation commands.

DATABASE FILES	- .DBF
MEMORY FILES	- .MEM
COMMAND FILES	- .CMD
REPORT FORM FILES	- .FRM
TEXT OUTPUT FILES	- .TXT
INDEX FILES	- .NDX
FORMAT FILES	- .FMT

Any legitimate CP/M filename may be used to refer to dBASE files. Remember, if, during an access of any file, the type is not supplied by the user, dBASE will assume the above file types. For further information regarding the use of filenames and types refer to the Digital Research publication "CP/M User's Guide".

3.1 DATABASE FILES (.DBF)

Databases are what dBASE is all about. dBASE's database files consist of a structure record and zero to 65535 data records. The structure record is essentially a map of the data record format. The structure can contain up to thirty-two different entries. Each entry in the structure refers to a field of data in the data records. The structure holds the following data:

* The name of the data fields
* The type of data within data fields
* The size of the data fields
* The position of the data within records

DATA FIELD NAME - The name may be up to 10 characters long. In all operations during a dBASE run the data fields will be referenced by this name. Field names are alphanumeric (plus colons) by nature. However, fields must begin with a letter and, colons must be embedded in the name. Some examples follow.

Examples of data field names:

```
        A
        A123456789
        ABC:DEF
        A:B:C:D:E
        ABCD:           invalid, colon not embedded
        ABC,DEF         invalid, comma is illegal
```

DATA TYPE - dBASE allows three types of data to be used to specify the contents of the data fields. They are: character strings ('ABCD'), numeric quantities (2 or 5*18), and logicals (true/false).

FIELD SIZE - This is the number of character positions (width) needed to contain the data that will be placed into this field. Character string fields and numeric fields may be from 1 to 254 positions in length. The count for a numeric field should include the decimal point. Logical fields are always one position in length. Also, for numeric fields, the number of positions to the right of the decimal point may also be contained in the structure.

Once the structure has been defined, the user can enter data values into the fields for as many records as are desired. Usually, there is only one structured data file available to the user at any given time (this is referred to as the USE file or the file in USE). There is however, a way to use two databases at one time. See the commands SELECT and JOIN.

3.2 MEMORY FILES (.MEM)

Memory files are static files of memory which are divided into variables similar to record variables. These variables are known as memory variables and are limited to 64 in number.

The values of memory variables are independent of the database in use. That is, the record position of the file in USE has no bearing on the variables in the memory file. Memory variables are used to contain constants, results of computations, and symbolic substitution strings (see Section 5), etc. The rules of naming, typing, and sizing of memory variables are identical to those of the field variables described above.

The SAVE command will write all current memory variables to a memory file; and the RESTORE command will read a saved memory file back into the memory variables.

3.3 COMMAND FILES (.CMD)

A command file contains a sequence of dBASE command statements. This provides the user with a method of saving a set of frequently used command sequences which then allows one to more easily manipulate database files.

Command files may be created and modified by text editors and/or word processors, although dBASE now has the capability to create/edit command files itself with the MODIFY COMMAND. Command files are started by the DO command. Command files may contain any dBASE commands, however, one should be careful since some of the commands (CREATE, INSERT, APPEND (from the keyboard)) require user inputs beyond the command file contents.

Command files may be nested, i.e. command files may contain DO commands which are then executed. Again, care should be exercised in that, dBASE allows, at most, 16 files to be open at any given time. Therefore, if there is a file in USE, only 15 command files may be nested. Certain commands also use work files (e.g. SORT uses 2 additional files; REPORT, INSERT, COPY, SAVE, RESTORE, and PACK use one additional file). For instance, if a SORT command is issued from the lowest command file in a nest, then only 13 levels of command file could be used (i.e. the USE file, 2 SORT work files and 13 command files = 16). Whenever a command file issues the RETURN command or whenever the end-of-file is encountered on a command file, the command file is closed and its resources are available for other commands.

3.4 REPORT FORM FILES (.FRM)

The REPORT command either generates a form file or uses an existing form file. The form file contains instructions to the report generator on titles, headings, totaling, and column contents. Form files are constructed by dBASE as part of the REPORT dialog. They can be modified by text editors or word processors, however, it is usually easier to define a new report form from the start.

3.5 TEXT OUTPUT FILE (.TXT)

The text output files are created when the "SET ALTERNATE TO <filename>" and "SET ALTERNATE ON" commands have been specified. See SET command for more details. Also, the COPY and APPEND commands assume a text (.TXT) file whenever the SDF (System Data Format) or DELIMITED options are used.

3.6 INDEX FILES (.NDX)

Index files are generated by the INDEX command of dBASE. They contain keys and pointers to records of a database file. Indexing is a dBASE technique that gives rapid location of data in a large database. See the INDEX command for more information.

3.7 FORMAT FILES (.FMT)

A format file contains only "@" statements and "*" comments. It is identified by the "SET FORMAT TO <filename>" command and is activated by subsequent READ commands. Like command files (which format files resemble), format files are created and modified by any good text processor or the MODIFY COMMAND capability. Format files are not, however, necessary. "@"'s and "*"'s statements are usually built into the command file that needs them.

4.0 EXPRESSIONS

An expression in dBASE is a group of simple items and operators that can be evaluated to form a new simple value. For example "2+2" is an expression that can be evaluated to the value "4". Expressions are not necessarily always numeric in nature. The expression 'abc'+'def' can be evaluated to the value 'abcdef' (character string concatenation), or the expression 1>2 can be evaluated to the logical (Boolean) value of ".F." (false).

Expressions in dBASE are formed from the following components:

* Database field variables
* Memory variables
* Constants within the commands (literals)
* Functions
* Operations

VARIABLES - A variable in dBASE is any data field whose value may change. The field names of the currently referenced record in a dBASE file are variables. Their contents may be changed by moving the file pointer or by editing the current record. Variablesare also created and changed by the commands, STORE, RESTORE, COUNT, SUM, WAIT, ACCEPT, or INPUT. These are called memory variables.

A variable may be one of three types:

* Character strings
* Numeric quantities
* Logicals

CONSTANTS - A constant (or literal) is a data item which has an invariant, self-defined value. For instance, 1, 'abc', and .T. are constants which have a constant value regardless of the position of the database or any memory variable commands. They are literals since they ARE the value they represent (as cpposed to variables which are names representing a value). The values they represent are, respectively: a numeric one, a character string (containing the letters "a", "b", and "c"), and a logical (Boolean) value of TRUE (".T.").

Character string constants must be enclosed in single quotes ('), double quotes ("), or in square brackets ([,]). If a character string contains one of these "delimiters", then it should be enclosed in a pair of one of the other ones. For example the strings 'abc[def]ghi' and [abc'def'ghi] are valid character strings while 'abc'def'ghi' is not.

Logical constants (true/false) are represented by "T", "t", "Y", or "y" for true values (denoting true or yes) and "F", "f", "N", or "n" for false values (denoting false or no).

4.1 FUNCTIONS

Functions are special purpose operations that may be used in expressions to perform things that are difficult or impossible using regular expressions. In dBASE, there are three basic types of functions: numeric, character, and logical. The function type is based on the type of value that functions generate.

INTEGER FUNCTION:

INT(<numeric expression>)

This function evaluates a numeric expression and discards the fractional part (if any) to yield an integer value. The value of the INT function is the truncated value of the numeric expression within.

Examples:

```
. ? INT(123.456)
  123
. STORE 123.456 TO X
  123.456
. ? INT(X)
  123
```

RECORD NUMBER FUNCTION:

#

The value of the record number function is the integer corresponding to the current record number.

Examples:

```
. ? #
  4     (assuming that a database is in USE and is positioned at
        record number 4)
. SKIP
. ? #
  5
```

STRING FUNCTION:

STR(<numeric expression>,<length>,[<decimals>])

This function evaluates a numeric expression and yields a character string. The value of the STR function is a character string of length <length>. If <decimals> is specified, it is the number of digits to the right of the decimal point. All specifiers may be literals, variables, or expressions.

CAUTION: When this function is used to generate a key for indexing, the specifiers MUST be literals.

Example:

```
? STR(123.456,9,3)
123.456
```

SUBSTRING FUNCTION:

$(<char expression>,<start>,<length>)

This function forms a character string from the specified part of another string. The value of the substring function is a character string of length <length> filled with characters from the character expression starting with character number <start> for <length> characters. <start> and <length> may be literals, variables or expressions.

If <length> is longer than the <char expression> or if between the <length> and <start> the <char expression> "runs out" of characters, then the result will be only those characters that are there. See the following examples.

CAUTION: When the function is used to generate a key for indexing, the specifiers MUST be literals.

Examples:

```
. ? $('abcdefghi',3,3)
  cde
. store 3 to m
  3
. store 3 to n
  3
. ? $('abcdefghi',m,n)
  cde
. ? $('abcdefghi',6,7)
  fghi
. DISPLAY FOR '8080'$TITLE
```

STRING TO NUMERIC FUNCTION:

VAL(<char string>)

This function forms an integer from a character string made of digits, signs, and up to one decimal point. The length of the integer is equal to the number of characters in the string. If the character string begins with numeric characters but has non numeric characters, then the value generated by the VAL function is the leading numeric characters.

Another way to convert character numbers into numerics is the use the "&" (see 5.0 Macros). The "&" will convert the string into a numeric (including the decimal) when the substitution is encountered.

Examples:

```
. ? VAL('123')
  123
. ? VAL('123xxx')
  123
. ? VAL('123.456)
   123
. STORE '123.456' TO NUM
123.456
. ? 14 + &NUM
  137.456
```

LENGTH FUNCTION:

LEN(<char string>)

This function yields an integer whose value is the number of characters in the named string.

Example:

```
. STORE 'abc' TO STRING
. ? LEN(STRING)
  3
```

DELETED RECORD FUNCTION:

*

This is a logical function which is .TRUE. if the current record has been marked for deletion, and .FALSE. otherwise.

Example:

```
. ? *
.T.           (assuming that a database is in USE and that its
                current record has been deleted using the DELETE
                command)
```

END-OF-FILE FUNCTION:

EOF

This is a logical function which is .TRUE. if the end of file has been reached for the file in USE (the current record will be the last record in the database).

Examples:

```
. ? EOF
.F.           (assuming that a database is in USE and is not
                positioned at the last record)
. GOTO BOTTOM
. ? EOF
.F.
. SKIP
. ? EOF
.T.
```

SUBSTRING SEARCH FUNCTION:

@(<char string 1>,<char string 2>)

This function yields an integer whose value is the character number in <char string 2> which begins a substring identical to <char string 1>. If string 1 does not occur in string 2 then the @ function will be of value zero. Note: the @ function is similar to the substring operator "$" except that it tells where the first string is found in the second string, and can well be pronounced "where is string 1 AT in string 2".

Example:

```
. ? @('def','abcdefghi')
  4
```

UPPER CASE FUNCTION

!(<char string expression>)

This function yields the same string as the character string expression except that all lower case characters are converted to upper case.

Example:

```
. ? !('abc')
ABC
```

NUMBER TO CHARACTER FUNCTION

CHR(<numeric expression>)

This function yields the ASCII character equivalent of the numeric expression. That is, if the expression were the number 13, then CHR(13) generates a carriage return ASCII character. This function is useful when the user needs to send direct controls to hardware devices, most often printers.

Example:

```
. ? 'abcd'+CHR(13)+'____'
abcd
```

DATE FUNCTION

DATE()

This function will generate a character string that contains the system date in the format MM/DD/YY. The character string always has a length of 8. Nothing goes between the parenthesis, they only indicate a function (to avoid problems with variables named "DATE".)

The dBASE system date can be entered at dBASE start-up time or at anytime using the SET DATE TO command.

Examples:

```
. ? DATE()
06/15/81
. STORE DATE() TO MEMVAR
06/15/81
. SET DATE TO 4 1 82
. ? DATE()
04/01/82
```

FILE FUNCTION

FILE(<string exp>)

This is a logical function which is .TRUE. if the <string exp> exists and is .FALSE. if it does not.

Example:

```
.? FILE('TRACE')
.T.
.USE TRACE
```

TYPE FUNCTION

TYPE(<exp>)

This function yields a one-character string that contains a 'C', 'N', or 'L' if the <exp> is of type Character, Numeric, or Logical respectively.

Example:

```
. STORE 1 TO X
. ? TYPE(X)
 N
```

TRIM FUNCTION

TRIM(<cstring>)

The TRIM function removes trailing blanks from a field. Usually dBASE carries trailing blanks on all variables to avoid column alignment problems on displays.

NOTE: This function must NOT be used in the INDEX command as the key length must be computable for internal dBASE usage.

Examples:

```
. STORE 'ABC   ' TO S
. ? LEN(S)
 6
. STORE TRIM(S) TO S
. ? LEN(S)
 3
```

4.2 OPERATIONS

There are four basic types of operations, arithmetic, comparison, logical and string. The specific operators in each class are listed below, and examples follow for the less familiar ones.

It is important to know that both "sides" of the operators must be the same type. That is, one may only add integers to integers or concatenate characters with characters, adding an integer to a character results in dBASE seeing a syntax error.

```
. STORE 3 TO A
  3
. STORE '3' TO B
3
. ? A+B

*** SYNTAX ERROR ***
   ?
? A+B
CORRECT AND RETRY(Y/N)?
```

This error occurs because numerics and characters are seen differently at the machine level; a numeric 3 is just that--3 hex, while a character 3 has the ASCII value of 33 hex. The program becomes confused, it does not know whether or not an addition is taking place or a concatenation. Using the same variables as in the previous example:

```
. ? A+VAL(B)
  6
```

The string '3' has been converted to an integer and the addition performed.

ARITHMETIC OPERATORS (generate arithmetic results)

+	= addition
-	= subtraction
*	= multiplication
/	= division
()	= parentheses for grouping

Examples:

```
. ? (4+2)*3                    An example of use of
 18                            arithmetic parentheses
. ? 4+(2*3)                    used for grouping
 10                            in calculations
```

COMPARISON OPERATORS (generate logical results)

<	= less than
>	= greater than
=	= equal
#	= not equal
<=	= less than or equal
>=	= greater than or equal
$	= substring operator (e.g. if A and B are character strings, A$B will be TRUE if and only if string A is equal to B, or is contained in B

Examples:

```
. ? 'abc'$'abcdefghi'               An example of the $
 .T.                                substring operator
. ? 'abcd'$'ghijkl'
 .F.
. DISPLAY FOR '8080'$TITLE          Results in all records with
                                    '8080' somewhere in the field
                                    TITLE being displayed on the
                                    screen
```

LOGICAL OPERATORS (generate logical results)

.OR.	= boolean or
.AND.	= boolean and
.NOT.	= boolean not (unary operator)

Examples:

```
. store t to a
 .T.
. store f to b
 .F.
. ? a .or. b
 .T.
. store .not. b to c
 .T.
. ? a .and. c
 .T.
```

STRING OPERATORS (generates string result)

+ = string concatenation
- = string concatenation with blank squash

Examples:

```
. STORE 'ABCD  'TO A
ABCD
. STORE 'EFGH' TO B
EFGH
. ? A+B
ABCD  EFGH
```

In a string concatenation the two strings are just appended to each other.

```
. STORE 'ABCDE   ' TO A
ABCDE
. STORE '1234  67' TO B
1234  67
. ? A-B
ABCDE1234  67
```

In a string concatenation with blank squash, the trailing blanks are moved to the end of the string. Leading and embeded blanks are not altered.

ORDER OF EXECUTION

The sets of operators for the arithmetic, string and logical have an order in which they are satisfied. That is, what operation is done before what other operations. The following table indicates the order of precedence for each of the three major operator classes. In each of the "levels" (1, 2, etc.) the order of execution is left-to-right.

Example:

```
. ? 4+2*3
 10
```

Arithmetic operator precedence	String operator precedence	Logical
1) parenthesis, functions	parenthesis, functions	.NOT.
2) unary +,-	relations, $(substring op)	.AND.
3) *,/	+,- (concatenation)	.OR.
4) +,-		
5) relations		

5.0 MACRO SUBSTITUTION

Whenever an ampersand (&) followed by the name of a character string memory variable is encountered in a command, dBASE replaces the & and memory variable name with the memory variable's character string. This allows the user to define some parts of a command once and call it out any number of times in various commands.

Macros are useful when complex expressions must be frequently used. They also allow parameter passing within command file nests. All characters between the ampersand and the next special character (including space) are taken as the memory variable name.

If the user desires to append characters to the symbolic substitution, then the memory variable name should be terminated with a period. The period will be removed like the ampersand at substitution time.

If an ampersand is not followed by a valid memory variable name then no expansion is attempted and the ampersand remains in the command line.

Examples:

```
. ACCEPT "Enter data disk drive letter" to DR
USE &DR:DATAFILE  (at execution time will be USE B:DATAFILE if
                   "B" was entered in response to the ACCEPT)


. STORE 'DELETE RECORD ' TO T
&T 5               (at execution time will be DELETE RECORD 5)
```

See appendix A for further examples.

6.0 INTERFACING WITH NON-dBASE PROCESSORS

dBASE can read data from files which were created by processors other than dBASE (e.g. BASIC, FORTRAN, PASCAL) and can generate files which can be accepted by other processors.

The APPEND command has the ability to read standard ASCII text files (using the CP/M convention of a line of text followed by a carriage return and line feed) by specifying the SDF (System Data Format) option. Similarly, the COPY command generates standard ASCII format files when the SDF option is used. Unless explicitly overridden, the file types of files created with the SDF and DELIMITED options will be .TXT.

Some processors and languages read and write files in a delimited format. In this form all fields are separated by commas and character strings are enclosed in quotes. dBASE can APPEND and COPY these files when the DELIMITED keyword is included in the command. If the DELIMITED feature is used, SDF is assumed.

Since some processors use single quotes and some use double quotes to delimit character strings, APPEND will accept either. The COPY command normally generates single quotes but will output any character as defined by the WITH phrase of the DELIMITED clause. It is strongly recommended that only single and double quotes be used.

A special case occurs when a "," is used in the WITH phrase for a COPY. All trailing blanks in character strings and leading blanks in numerics are trimmed. Also, character strings will not be enclosed with quotes or any other character.
M

Examples:

```
.USE <FILENAME>.DBF
.COPY TO <FILENAME>.TXT DELIMITED WITH "

.USE <FILENAME>.DBF
.APPEND FROM <FILENAME>.DAT SDF
```

7.0 CLASSES OF COMMANDS

During the normal use of dBASE, various commands are used in combination to accomplish a particular task. Such groups are shown below. Some dBASE commands are patterned after the structured constructs that most "modern" computer languages use. These commands are in the COMMAND FILE class of commands. There are some special rules that control the use of these commands, which are expounded upon in section 9.0.

CREATION OF FILES - the following commands create database files and associated files:

```
* CREATE - create new structured database files
* COPY   - copy existing databases to create copies
* MODIFY - alters database structures
* REPORT - create a report form file
* SAVE   - copy the memory variables to mass storage
* INDEX  - creates an index file
* REINDEX  realigns an old index file
* JOIN   - outputs the JOIN of two databases
* TOTAL  - outputs a database of totalled records
```

ADDITION OF DATA - the following commands add new data records to databases:

```
* APPEND - add data at end of a file
* CREATE - allows addition of data at creation
* INSERT - insert data into a file
```

EDITING OF DATA - the following commands edit the data within a database:

```
* CHANGE  - edit columns of fields
* BROWSE  - full screen window viewing and editing
* DELETE  - marks records for deletion
* EDIT    - alter specific data fields in a database
* PACK    - removes records marked for deletion
* RECALL  - erases mark for deletion
* REPLACE - replaces data fields with values
* READ    - replaces data from user defined full-screen
* UPDATE  - allows batch updates of a database
```

DATA DISPLAYING COMMANDS - the following commands display selected data from a database:

* @ - displays user formated data on CRT or printer
* BROWSE - displays up to 19 records with as many fields as will fit on the screen
* COUNT - count the number of records that meet some conditional expression
* DISPLAY - displays records, fields, and expressions
* READ - displays data and prompting information in full-screen mode
* REPORT - format and display a report of data
* SUM - compute and display the sum of an expression over a group of database records
* ? - displays an expression list

POSITIONING COMMANDS - the following commands position the current record pointer to records as directed:

* CONTINUE- positions to next record with conditions specified in the LOCATE command
* FIND - positions to record corresponding to a key on indexed files
* GOTO - position to a specific record
* LOCATE - find a record that fits a condition
* SKIP - position forwards or backwards

FILE MANIPULATING COMMANDS - the following commands affect entire database files:

* APPEND - append dBASE files or files in System Data Format (SDF)
* COPY - copy databases to other databases or SDF files
* DELETE - delete files
* DO - specifies a command file from which subsequent commands are to be taken
* RENAME - rename a file
* SELECT - switches between USE file
* SORT - create a copy of a database which is sorted on one of the data fields
* USE - specifies the database file to be used for all operations until another USE is issued

MEMORY VARIABLE COMMANDS - the following commands manipulate the memory variables:

* ACCEPT - stores a char string into memory variables
* COUNT - stores counts into memory variables
* DISPLAY - can display memory variables
* INPUT - stores expressions into memory variables
* RESTORE - retrieves sets of stored memory variables
* SAVE - save the memory variables to a file
* STORE - stores expressions into memory variables
* SUM - stores sums into memory variables
* WAIT - accepts a single keystroke into a memory variable

COMMAND FILE COMMANDS - the following commands assist in the control and usage of command files:

* ACCEPT - allows input of character strings into memory variables
* CANCEL - cancels command file execution
* DO - causes command files to be executed and allows structured loops in command files
* IF - allows conditional execution of commands
* ELSE - alternate path of command execution within IF
* ENDDO - terminator for DO WHILE command
* ENDIF - terminator for IF command
* INPUT - allows input of expressions into memory variables
* LOOP - skips to beginning of DO WHILE
* MODIFY COMMAND - allows editing of command files
* RETURN - ends a command file
* SET - sets dBASE control parameters
* WAIT - suspends command file processing

DEVICE CONTROLLING COMMANDS - the following commands control peripheral devices like printers and CRT's:

* EJECT - ejects a page on the list device
* ERASE - clears the CRT

8.0 FULL SCREEN OPERATION

The following are cursor control keys for full screen operation:

ctl-E,A	-	Backs up to previous data field.
ctl-X,F	-	Advances to next data field.
ctl-S	-	Backs up one character in data field.
ctl-D	-	Advances one character in data field.
ctl-Y	-	Clears out current field to blanks.
ctl-V	-	Switches (toggles) between overwrite and insert modes.
ctl-G	-	Deletes character under cursor.
RUBOUT	-	Deletes character to left of cursor.
ctl-Q	-	Aborts full screen and returns to normal dBASE control. Changes to database variables are abandoned.

When in EDIT:

ctl-U	-	Switches (toggles) the current record between being marked for deletion and unmarked.
ctl-R	-	Writes current record back to disk and displays previous record i.e. backs up a record.
ctl-C	-	Writes current record back to disk and displays next record i.e. advances to next record.
ctl-W or ctl-O	-	Writes current record to disk and exits screen edit mode. (ctl-O is for Superbrain)

When in MODIFY

ctl-N	-	Moves all items down one to make room for an insertion of a new field.
ctl-T	-	Deletes the field where the cursor is and moves all lower fields up.
ctl-C	-	Scrolls fields down.
ctl-R	-	Scrolls fields up.
ctl-W or ctl-O	-	Writes data to the disk and resumes normal operations. (ctl-O is for Superbrain).
ctl-Q	-	Exits without saving changes.

When in APPEND, CREATE, or INSERT:

ctl-C or ctl-R	-	Write current record to disk and proceed to next record.

Carriage return when no changes have been made and cursor is in initial position - terminate operation and resume normal dBASE operations.

When in BROWSE:

ctl-U	-	Switches (toggles) the current record between being marked for deletion and unmarked.
ctl-R	-	Writes current record back to disk and displays previous record i.e. backs up a record.
ctl-C	-	Writes current record back to disk and displays next record i.e. advances to next record.
ctl-W or ctl-O	-	Writes current record to disk and exits screen edit mode. (ctl-O is for Superbrain)
ctl-Z	-	Pans the window left one field.
ctl-B	-	Pans the window right one field.

9.0 COMMANDS

The explicit definitions of the dBASE commands are in this section. The user should familiarize him/herself with these fundamentals before reading the rest of the command information.

9.1 SYMBOL DEFINITIONS

Understanding what the special symbols in the general formats of the dBASE commands really mean is vitally important. Not only does it help in understanding just what the form of the command really, it helps to show the potential of each command. Please read the following table throughly.

Symbol	Meaning
<commands> or <statements>	- means any valid dBASE statements; it also means whole statements. An IF without an ENDIF, (or a DO WHILE without an ENDDO), is only half of a statement, while a REPORT is a whole statement in itself.
<char string> or <cstring>	- means any character string; character strings are those characters that are enclosed in single quotes ('), double quotes ("), or square brackets ([]).
<delimiter>	- means any special character; special characters are those characters from the keyboard that are punctuation marks, like any one of the following "()*=,@.
<exp>	- means an expression; an expression can be created by tacking together numbers, functions, field names or character strings in any meaningful manner. "4+8", and "doc = '3' .or. doc = '4'", are both expressions as well as "$('abc'+&somestr,n,3) = 'abcdefg'".
<exp list>	- means a list of expressions separated by commas; usually simple expressions are used. Two of the examples in the previous paragraph are rather complicated, the first one could be considered as simple.
<field>	- means any record field name; in one of the examples that are in the following commands, one of the databases has field names like ITEM, COST, DATE, etc.
<field list> or <list>	- means a list of record field names separated by commas.

<file> or <file name>	- means any filename; these are file names that must obey the rules for file names that were stated in section 3.0.
<form file>	- means the name of a report form filename; see section 3.4 and the REPORT command for the how and why of this type of file.
<index file>	- means the name of the file where indexing information is placed; see section 3.6 and the INDEX command for the how and why of this type of file.
<key>	- means the field name which will be indexed on; keys are important. There may be several indexes for any given database, each on different (or on a combination of).keys. Keys may be <expressions> or field names. See the INDEX command for more information.
<memvar>	- means any memory variable; memory variables are those variables that are created by STOREs or by use of a command that saves some value for later use (ACCEPT, INPUT, etc.) There is a maximum of 64 memory variables allowed in dBASE.
<memvar list>	- means a list of memory variables separated by commas.
<n>	- means a literal; literals are numbers which are not gotten from memory variables or calculations."4+3" is not a literal, while "4" and "9876" are literals.
<scope>	- means a specification of the scope of the command; scope means how much does the command cover. There are three values that <scope> may take on.
ALL	- means all the records in the file. ALL means that the file is rewound and whatever the command ALL the records in the file are searched for compliance. ALL is the default for some of the commands. For other commands the default will be the current record (specially for the more potentially destructive commands like DELETE). Each command description tell what is the default scope. In the case of using a FOR phrase in any of the commands, ALL will be the default.
NEXT n	- means the next n records, including the current record; NEXT also begins with the record currently being pointed at. And n must have a literal value, that is, it must not be a memory variable or an expression.
RECORD n	- means only record n; again, n must not be a memory variable or an expression--it must be literal before it will work.
FOR <exp>	- Any record so long as some logical expression has a true value. Unless otherwise specified, the presence of a FOR clause causes ALL records to scanned (with a rewind of the database).
WHILE <exp>	- All sequential records as long as some

logical expression (<exp>) has a true value. The controlling command stops the first time the expression is false. The presence of a WHILE clause implies NEXT 65534 unless otherwise specified and does not rewind the database.

There are other special symbols used in the command formats. These are special to the command and will be explained in the body of the command.

9.2 RULES TO OPERATE BY

As with all command "languages" there are a set of rules which must be followed to successfully operate the program. The following rules are to use in translating the general format of the commands into the more useful specific forms.

1. The verb of any command must be the first non-blank character of the command line; the phrases may follow in any order. A verb is an action word; CREATE, APPEND, REPORT, SET, DISPLAY, and ERASE are all examples of verbs--they cause a specific action. Phrases are equivalent to adverbs; they more fully describe the action. FOR, NEXT, and WITH are examples of words that begin phrases. All of these example words are refered to as "keywords".

2. Any number of blanks may be used to separate words and phrases. Remember though, blanks are counted in the 254 limit described in Rule #3.

3. All commands must be less than 254 characters in length (even after a macro expansion).

4. Commands and keywords can be abbreviated to the first four (or more) characters. E.g. DISPLAY STRUCTURE could be input as DISP STRU or DISPL STRUCT or etc. Just remember that the abbreviation must also be spelled correctly up to the point where it ends.

5. Either upper or lower case letters may be used to enter commands, keywords, field names, memory variable names, or file names.

6. Parts of the commands are optional, that is, some parts of the commands may be left off when the command is used. Square brackets ([]) are used in the command formats to show which phrases are the optional constructs that may be left off. These are the phrases which are used to modify the action of commands. The upper case words are the keywords and they must be entered whenever the phrase that contains them is used.

7. A reserved word is a keyword that will generate an error if is

7. A reserved word is a keyword that will generate an error if is used for something other that what it is supposed to be. There are no reserved words in dBASE. However, certain field names and file names can cause difficulty, e.g., a command file named WHILE will be incorrectly interpreted as a DO WHILE statement by the DO command processor, ALL as a field name cannot be used in a number of commands. In general, it is a good practice to avoid the use of dBASE keywords as field names or file names.

8. dBASE statements in a command file must nest correctly. To nest something means that one statement must fit inside another statement. This is especially important to proper execution of the IF-ELSE-ENDIF and the DO WHILE-ENDDO groups. Indenting a command file will show if the statements are correctly nested. dBASE does not catch nesting errors, it will however execute the command file in an unknown manner. Below are examples of how to correctly nest these two statements.

```
DO WHILE .NOT. EOF
  .
  statements
  .
  IF A .AND. B
    .
    more statements
    .
  ELSE
    .
    DO WHILE A <= 57
      .
      some more statements
      .
    ENDDO
    .
    even more statements
    .
  ENDIF
  .
  infinitely more statements
  .
ENDDO
```

This is the correct way to nest. The IF-ELSE-ENDIF statement is totally within the DO WHILE-ENDDO statement. Just as the second DO WHILE-ENDDO statement is totally within the ELSE part of the IF-ELSE-ENDIF. It would be just as easy to show more levels of nesting, since dBASE allows many more levels to exist.

```
DO WHILE .NOT. EOF
  . .
  statements
  .
  IF something changes values
  .
  ENDDO
  .
  more statements
  .
  ENDIF
```

This is an example of a NO NO. The ENDDO crossed over the boundary of the IF-ENDIF group, that is, the two statements do not nest properly. The command file that holds these statements will not work as expected AND dBASE will not explain why.

?

```
? [<exp list>]
?? [<exp list>]
```

This command is a specialized form of the DISPLAY command; it is equivalent to DISPLAY OFF <exp>. It can be used to show the value an expression or list of expressions. The question mark command (possibly pronounced "what is" can use memory variables, database fields, constants, or functions. A "?" with no expression spaces down a line on the output. This feature is particularly useful in command files to "open up" the displays.

The second form of this command "??" behaves like a single "?" except that no line feed or carriage return is done before the expression is printed. This can be used in command files to output more than one expression to the same output line.

Examples:

```
. USE EXAMPLE
. 4
. ? #
 4
. ? NAME
CHANG, LEE
. ? 5+9
 14
```

Following is a sample command file that uses the ? to space out the display. The command file is set up to be executed with the command: "DBASE H:FILE". The dBASE response to the command file follows the command file.

```
set default to g
use trace index trace
disp stru
?
accept "Enter today's date." to dte
set date to &dte
release dte
return
```

```
STRUCTURE FOR FILE:  TRACE.DBF
NUMBER OF RECORDS:   02359
DATE OF LAST UPDATE: 10/08/81
PRIMARY USE DATABASE
FLD      NAME       TYPE WIDTH   DEC
001      UP          C    024
002      TRFLD       C    005
003      DOC         C    024
004      DESCR       C    080
005      NATURE      C    010
006      STATUS      C    006
007      TESTED      C    004
** TOTAL **              00154

Enter today's date.:10 14 81
```

```
@ <coordinates> [SAY <exp> [USING <format>]]
                [GET <variable> [PICTURE <format>]]
```

This command works with the SET FORMAT TO, ERASE, EJECT, CLEAR GETS and READ commands and is a most powerful way to display specific, formatted information on the screen or the printer. The way an "@" is interpreted changes according to how the SET FORMAT TO command is used. Also whether or not one of the other commands has an effect also depends on the SET command. All combinations are discussed below.

The <coordinates> are an "x,y" pair and may take on one of two meanings, either they are screen coordinates or they are printer coordinates. The "x,y" denotes line (x) and column (y). On most CRTs, the screen oriented coordinates have an "x" range of 0-23, and a "y" range of 0-79, that is 24 lines by 80 columns. dBASE uses the 0th line for messages to the user and the user should avoid using it. The printer oriented coordinates have both an "x" and a "y" range of 0-254. For either of these two meanings the coordinates can be any literal, numeric memory variable, or numeric expression. The SET FORMAT command is used to choose between either of these two meanings.

When a SET FORMAT TO SCREEN command has been issued (which is the default), the "@" command causes data to be displayed on the screen. A coordinate pair of 0,0 means the first character location on the upper left corner of the display. (This frequently referred to as the home position.) The pair 10,15 means the 11th line and the 16th column of the display. Again the 0th line on the screen should not be used. "@" commands may be issued in any order to the screen. That is, one may SAY something to line 15 before one SAYs something to line 10. Likewise columns may be filled in any order.

When a SET FORMAT TO PRINT command had been issued, the "@" command will cause data to be printed on the printer. The coordinate pair 0,0 refers to the upper left hand corner of the paper. "@" commands to the printer must be output in order. Much paper will be wasted if this is not done. The user may like to pretend that a typewriter is being use (indeed, it is). All commands to line 5 must preceed commands to line 6, also, all commands to column 10 must preceed commands to column 20, etc. I this is not done a page eject will occur before the new line is printed.

When the SET FORMAT TO SCREEN has been issued, an ERASE will clear the screen of all information that was previously on it, will release all the GETs (see below), and will reset the coordinates to 0,0. When the SET FORMAT TO PRINT has been issued an EJECT will do a page feed and reset the coordinates to 0,0.

The SAY phrase is used to display an expression that will not be altered by subsequent editing via the READ command. The USING subphrase is used to format the expression emitted by the SAY phrase. Formatting directives are explained below. It is a good thing to always use the USING subphrase. dBASE will take liberties with the expression if there is no USING.

SAY phrases may be used on either the screen or the printer. GETs however, will only be recognized when the SET FORMAT TO SCREEN command has been issued.

The GET phrase displays the current value of a field variable or memory variable. The variable must exist prior to issuing of the GET and is subject to later editing by the READ command. The PICTURE phrase may be used with a GET phrase to allow special formatting and validation of the data as it is entered (see the READ command for further information). If no PICTURE clause is given, then the data type (character, numeric or logical) forms an implicit PICTURE.

If the data type of the field variable or memory variable in the GET is logical then the data validation allows only the characters 'T', 'F', 'Y', 'N' and their lower case equivalents to be entered.

A maximum of 64 GETs can be active at any given time. Either the ERASE command or the CLEAR GETS command may be used to release the existing GETs.

When SET FORMAT TO SCREEN is in effect and if neither a SAY or a GET phrase is given, then the remainder of the line indicated by the coordinates is cleared to spaces. Thus @ 10,0 will clear the entire 11th line.

When the SET FORMAT TO SCREEN is in effect, a READ must be issued in order to "fill" the GETs. (See the READ command). However when SET FORMAT TO PRINT is in effect, "@" commands require no subsequent READ commands to complete their action.

Not needing a READ to print allows the user to directly format the output for any pre-printed material (such as checks, purchase orders, etc.) in a most convenient manner. The user need only to remember that "@" commands must be issued as if one were typing on a typewriter.

In using the SET FORMAT TO PRINT capability, it is often necessary to print out more than one item. The ability to subsitute memory variables for the coordinate values is important. The following example is from a command file that generates a special report form for a special task.

```
SET FORMAT TO PRINT
GOTO TOP
STORE 7 TO CNTR
DO WHILE .NOT. EOF
   IF CNTR >= 50
      EJECT
      STORE 7 TO CNTR
   ENDIF
   @ CNTR,12 SAY P USING 'XXXXXXXXXXXXXXXXXX XX..  XXXXXX'
   @ CNTR,48 SAY D USING 'XXXXXXXXXX'
   @ CNTR,64 SAY P1 USING 'XXXXXXXXXXXXXXXXXX
   @ CNTR,88 SAY U USING 'XXXXXXXXXX'
   @ CNTR,104 SAY P2 USING 'XXXXXXXXXXXXXXXXX) '
   IF RCD <> 0
      @ CNTR,130 SAY RCD USING '9999'
   ENDIF
   STORE CNTR + 1 TO CNTR
   SKIP
ENDDO
RETURN
```

In this command file, a maximum of 57 lines will be printed on the printer before a page eject is done. The purpose here was to print out most of the fields of a database (and selectively print out one of the fields). Care must be taken to make sure enough room is given to the SAY phrase to emit the variable. If the USING is shorter than the variable or the field, the variable or field is truncated. The <format> for the USING (the 'XXX...X' strings are explained in the table below.

Also, in the SET FORMAT TO PRINT mode, if the coordinates of the next "@" allow information to be printed on the same line but start it in a column that has already been printed, the printer may not output the proper information. In fact, the printer may go to the extreme right and print (in one square) all the information in the rest of the line. In the SET FORMAT TO SCREEN mode, the old information will be written over by the new information.

The last form of the SET FORMAT command is: SET FORMAT TO <format file>. When this command is in effect and when a READ command has been issued, the "@" commands are READ from the pre-designed <format file>. In this manner the user may design the screen into a format for more specialized purposes. It is important to note here that the use of format files is not necessary for use of "@"s, since "@"s may reside in command files. See READ for more information.

Formats:

Both the USING and PICTURE clauses have as their object, a format. The format is a series of characters that indicate which characters appear on the screen or page. The following table defines the characters and their functions:

Format character	SAY function	GET function
	causes the next number to be output	allows only a digit (1,2,...,8,9,0) and the characters ".", "+", "-", and " " (a space) to be entered
9	same as #	same as #
X	outputs the next character	allows any character to be entered
A	outputs the next character	allows only alpha. to be entered
$ or *	outputs either a digit or a $ or * instead of leading zeros	output as is
	no effect	converts lowercase alpha characters to uppercase

Example:

. @ 5,1 SAY 'ENTER PHONE NUMBER' GET PNO PICTURE '(999)999-9999'

The message 'ENTER PHONE NUMBER' would be displayed, followed by '(bbb)bbbb-bbbb' (b indicates a blank) assuming that the value of PNO was all blanks prior to issuance. When (and if) the READ command is issued, only digits can be entered. The value of PNO after the READ command might well be '(213)555-5555' after editing. All of the non-functional characters in the PICTURE format are inserted into the variable. In this example, the parentheses, minus sign and the blank are non-functional.

. @ 10,50 SAY HOURS*RATE USING '$$$$$$$.99'

This "@" command could be used with either the screen or the printer since it has no GET phrase. It might well be used to print payroll checks. The dollar signs will be printed as long as there are leading zeros in the item to be printed. If hours=40 and rate = 12.50 then '$$$$500.00' will be displayed. This feature is known as floating dollar and is valuable for printing checks that cannot be easily altered in value.

When commas are used in the integer part of a picture, they are replaced by the picture character in front of them if there are no significant digits in the item to the left of where the comma would otherwise be placed.

. @ 10,50 SAY HOURS * RATE USING '$$$,$$$.99'

Would output $$$$500.00 and specifically not output $$$,500.00.

Normally, a number of "@" commands are issued then, if any GET phrases were included, a READ command is issued to allow editing or data entry into the GET variables. In the following example the screen is formatted with several "@"s and a database is filled with information according to these "@"s. The last record in the database will have a "0" in the field "name", this is the record that will be deleted, since it is not necessary.

```
SET FORMAT TO SCREEN
USE F:EXAMPLE
ERASE
DO WHILE NAME # '0'
  APPEND BLANK
  @ 5,0 SAY "ENTER NEXT NAME" ;
       GET NAME PICTURE 'XXXXXXXXXXXXXXXXXXXX'
  @ 6,0 SAY "ENTER TELEPHONE NUMBER";
       GET TELE:EXTSN PICTURE 'XXXXX'
  @ 6,40 SAY "ENTER MAIL STOP" ;
       GET MAIL:STOP PICTURE 'XXXXXXXXXX'
  READ
ENDDO
GOTO BOTTOM
DELETE
PACK
LIST
RETURN
```

The following commands affect the operation of the "@" command:

- **SET INTENSITY ON/OFF (default is ON) affects the screen intensity of GET's and SAY's.**
- **SET BELL ON/OFF (default is ON) affects the bell alarm when invalid characters are entered or a data boundary is crossed.**
- **SET COLON ON/OFF (default is ON) affects whether GET variables are bounded by colons.**
- **SET DEBUG ON/OFF (default is OFF) allows easier debugging of "@" commands by shifting ECHO and STEP messages to the printer.**
- **SET SCREEN ON/OFF (default is ON) allows use of full screen operations.**
- **SET FORMAT TO SCREEN/PRINT/<format file> determines device destination of output (SCREEN or PRINTer). SET FORMAT TO <format file> establishes a format file as the source of "@" commands for the READ command. SCREEN is the default value.**
- **READ enters the editing mode so that GET variables can be altered.**

ACCEPT

ACCEPT ["<cstring>"] TO <memvar>

This construct permits the entry of character strings into memory variables just as the INPUT command, but without the necessity of enclosing them in the quote marks required by the INPUT command. ACCEPT makes a memory variable of the type 'character' out of whatever is entered; INPUT determines the data type from the syntax of the entry and makes a memory variable of that type.

The <memvar> is created ,if necessary, and the input character string is stored into <memvar>. If "<cstring>" is present, it is displayed on the screen, followed by a colon, as a prompt message before the input is accepted. If a carriage return is entered in response to an ACCEPT request, <memvar> will receive a single space character. Either single quotes, double quotes, or square brackets may be used to delimit the prompt string, however, both the beginning and ending marks must correspond.

Examples:

```
. ACCEPT "ENTER PERSONS NAME" TO NAM
ENTER PERSONS NAME:John Jones

. ACCEPT "ENTER PERSON'S NAME" TO NAM2
ENTER PERSON'S NAME:Dave Smith

. DISP MEMO
NAM        (C)  John Jones
NAM2       (C)  Dave Smith
** TOTAL **      02 VARIABLES USED   00020 BYTES USED

. ACCEPT TO ANY
:ANY CHARACTERS

. DISP MEMO
NAM             John Jones
NAM2            Dave Smith
ANY             ANY CHARACTERS
** TOTAL **      03 VARIABLES USED   00034 BYTES USED
```

APPEND
------

a. APPEND FROM <file> [FOR <exp>] [SDF] [DELIMITED WITH <delimiter>]
b. APPEND BLANK
c. APPEND

In all three forms, records are appended onto the database in USE. APPEND, CREATE, and INSERT are the only commands that allow the addition of records to a database. APPEND and CREATE allow multiple additions at one time, INSERT allows only one.

In the first form, the records to be appended are taken from another file, i.e. <file>. If the SDF clause is present, the records are assumed to be in System Data Format (see section 6.0). If the new records are smaller than the old records in the USE file, then the new record is padded on the right side with blanks; if the new records are longer then the USE file records, then the newly appended records are truncated. Records are added to the USE file until end-of-file is detected upon the FROM file.

If the DELIMITED keyword is in the APPEND command, then the records taken from the FROM file are assumed to be delimited and appended accordingly. Many computer languages generate files where character strings are enclosed in single or double quotes and fields are separated by commas. In the delimited mode, dBASE removes the quotes and commas from delimited files and stores the data into a dBASE-structured database, according to the database's structure.

If the SDF and DELIMITED clauses are not present, then the FROM file is assumed to be a dBASE-structured database file. The structures of the USE and FROM file are compared. Fields which occur in the records of both files are taken from the FROM file and appended onto the USE file. Padding and truncation are performed as appropriate to force the FROM data items into the USE file's structure.

If the FOR phrase is used, then dBASE appends the records in the FROM <file> one by one, each time checking to see if the condition in the FOR is true. That is, the first record is appended. If the expression is true then the record is kept and dBASE will skip on to the next record. If the expression then the record is discarded and dBASE will again skip on to the next record. This procedure will continue until the end-of-file is reached for the FROM <file>. The implications of this is that the fields used in the expression must reside in the file receiving the new records.

If the BLANK clause (form b) is specified, a single, space filled record is appended to the USE file. This record can then be filled by the EDIT or REPLACE statements.

If no clauses follow the APPEND command (form c.), the user is prompted with the field names from the USE file's structure. Any number of new records may be created from the keyboard. The append mode is terminated when a carriage return is entered as the first character of the first field.

If the database in USE is an indexed database then the index file specified in the USE command is automatically updated when the new records are appended (except for APPEND BLANKs). Any other index file associated with that database must be re-indexed.

When APPENDing in the full-screen mode, the SET CARRY ON command will cause all of the data from the previous record to be carried over to the next record. Changes can then be made. This is especially useful if sucessive records have a lot of common data.

The APPEND command is especially useful when it is necessary to expand/contract fields or add/delete fields from an existing database. Using the CREATE command, set up a new database containing the desired structure and then APPEND the old database to the new. Fields which appear only in the new database will be blank filled.

Examples:

```
. USE EXAMPLE

. DISPLAY STRUCTURE
STRUCTURE FOR FILE:   EXAMPLE
NUMBER OF RECORDS:    00005
DATE OF LAST UPDATE:  12/31/80
PRIMARY USE DATABASE
FLD       NAME       TYPE WIDTH   DEC
001       NAME        C    020
002       TELE:EXTSN  C    005
003       MAIL:STOP   C    010
** TOTAL **               00036

. DISPLAY ALL
00001  NEUMAN, ALFRED E.     1357  123/456
00002  RODGERS, ROY          2468  180/103
00003  CASSIDY, BUTCH        3344  264/401
00004  CHANG, LEE            6743  190/901
00005  POST, WILEY           1011  84/13B
```

```
. APPEND

RECORD 00006

NAME:       LANCASTER, WILLIAM J
TELE:EXTSN: 6623
MAIL:STOP:  170/430

RECORD 00007

NAME:       NORRIS, R. "BOB"
TELE:EXTSN: 8093
MAIL:STOP:  427/396

RECORD 00008

NAME:      (cr)

. DISPLAY ALL OFF NAME,TELE:EXTSN
NEUMAN, ALFRED E.      1357
RODGERS, ROY           2468
CASSIDY, BUTCH         3344
CHANG, LEE             6743
POST, WILEY            1011
LANCASTER, WILLIAM J   6623
NORRIS, R. "BOB"       8093

. APPEND FROM DUPE3
00007 RECORDS ADDED

. DISPLAY ALL
00001  NEUMAN, ALFRED E.      1357  123/456
00002  RODGERS, ROY           2468  180/103
00003  CASSIDY, BUTCH         3344  264/401
00004  CHANG, LEE             6743  190/901
00005  POST, WILEY            1011  84/13B
00006  LANCASTER, WILLIAM J   6623  170/430
00007  NORRIS, R. "BOB"       8093  427/396
00008  NEUMAN, ALFRED E.      1357
00009  RODGERS, ROY           2468
00010  CASSIDY, BUTCH         3344
00011  CHANG, LEE             6743
00012  POST, WILEY            1011
00013  LANCASTER, WILLIAM J   6623
00014  NORRIS, R. "BOB"       8093

. APPEND BLANK

. DISPLAY
00015

. REPLACE NAME WITH 'RINEHART, RALPH'
00001 REPLACEMENT(S)
```

```
. DISPLAY
00015  RINEHART, RALPH

. DISPLAY ALL NAME,' ex =',TELE:EXTSN
00001  NEUMAN, ALFRED E.      ex = 1357
00002  RODGERS, ROY           ex = 2468
00003  CASSIDY, BUTCH         ex = 3344
00004  CHANG, LEE             ex = 6743
00005  POST, WILEY            ex = 1011
00006  LANCASTER, WILLIAM J   ex = 6623
00007  NORRIS, R. "BOB"       ex = 8093
00008  NEUMAN, ALFRED E.      ex = 1357
00009  RODGERS, ROY           ex = 2468
00010  CASSIDY, BUTCH         ex = 3344
00011  CHANG, LEE             ex = 6743
00012  POST, WILEY            ex = 1011
00013  LANCASTER, WILLIAM J   ex = 6623
00014  NORRIS, R. "BOB"       ex = 8093
00015  RINEHART, RALPH        ex =

. USE B:SHOPLIST

. DISP STRU
STRUCTURE FOR FILE:    B:SHOPLIST.DBF
NUMBER OF RECORDS:     00009
DATE OF LAST UPDATE:   06/22/79
PRIMARY USE DATABASE
FLD      NAME       TYPE WIDTH   DEC
001      ITEM        C    020
002      NO          N    005
003      COST        N    010    002
** TOTAL **              00036

. CREATE
FILENAME: NEWSHOP
ENTER RECORD STRUCTURE AS FOLLOWS:
 FIELD     NAME,TYPE,WIDTH,DECIMAL PLACES
 001       ITEM,C,25
 002       NO,N,5
 003       COST,N,10,2
 004       NEED:DATE,C,8
 005       (cr)
INPUT NOW? N

. USE NEWSHOP
. APPEND FROM B:SHOPLIST
00009 RECORDS ADDED

. LIST
00001  BEANS                     5      0.75
00002  BREAD LOAVES              2      0.97
00003  T-BONE                    4      3.94
00004  PAPER PLATES              1      0.86
00005  PLASTIC FORKS             5      0.42
00006  LETTUCE                   2      0.53
00007  BLEU CHEESE               1      1.96
00008  MILK                      2      1.30
00009  CHARCOAL                  2      0.75

. REPLACE ALL NEED:DATE WITH ' 7/ 4/76'
00009 REPLACEMENT(S)

. LIST
00001  BEANS                     5      0.75  7/ 4/76
00002  BREAD LOAVES              2      0.97  7/ 4/76
00003  T-BONE                    4      3.94  7/ 4/76
00004  PAPER PLATES              1      0.86  7/ 4/76
00005  PLASTIC FORKS             5      0.42  7/ 4/76
00006  LETTUCE                   2      0.53  7/ 4/76
00007  BLEU CHEESE               1      1.96  7/ 4/76
00008  MILK                      2      1.30  7/ 4/76
00009  CHARCOAL                  2      0.75  7/ 4/76
```

(The following example demonstrates the DELIMITED file append. This file could have been created by a number of different versions of BASIC)

```
'BARNETT, WALT',31415,6
'NICHOLS, BILL',76767,17
'MURRAY, CAROL',89793,4
'WARD, CHARLES A.',92653,15
'ANDERSON, JAMES REGINALD III','11528',   16
```

(Append the file into a dBASE-structured database)

```
. USE ORDERS

. DISP STRU
STRUCTURE FOR FILE:    ORDERS.DBF
NUMBER OF RECORDS:     00008
DATE OF LAST UPDATE:   00/00/00
PRIMARY USE DATABASE
FLD        NAME      TYPE WIDTH   DEC
001      CUSTOMER     C    020
002      PART:NO      C    005
003      AMOUNT       N    005
** TOTAL **              00031

. LIST
00001  SWARTZ, JOE           31415     13
```

2

```
00002  SWARTZ, JOE           76767     13
00003  HARRIS, ARNOLD        11528     44
00004  ADAMS, JEAN           89793     12
00005  MACK, JAY             31415      3
00006  TERRY, HANS           76767      5
00007  JUAN, DON             21828      5
00008  SALT, CLARA           70296      9

. APPEND FROM DELIM.DAT DELIMITED
00005 RECORDS ADDED

. LIST
00001  SWARTZ, JOE           31415     13
00002  SWARTZ, JOE           76767     13
00003  HARRIS, ARNOLD        11528     44
00004  ADAMS, JEAN           89793     12
00005  MACK, JAY             31415      3
00006  TERRY, HANS           76767      5
00007  JUAN, DON             21828      5
00008  SALT, CLARA           70296      9
00009  BARNETT, WALT         31415      6
00010  NICHOLS, BILL         76767     17
00011  MURRAY, CAROL         89793      4
00012  WARD, CHARLES A.      92653     15
00013  ANDERSON, JAMES REGI  11528     16
```

(The following examples demonstrates an APPEND FROM <file> FOR <exp>. Note that the fields in the FOR are in the USE file also.)

```
. USE CHECKS
. DISP STRU
STRUCTURE FOR FILE:   CHECKS.DBF
NUMBER OF RECORDS:    00013
DATE OF LAST UPDATE:  10/18/81
PRIMARY USE DATABASE
FLD       NAME       TYPE WIDTH   DEC
001     NUMBER        N    005
002     RECIPIENT     C    020
003     AMOUNT        N    010    002
004     HOME          L    001
005     OUTGOING      L    001
** TOTAL **              00038
```

```
. LIST
00001      1 Phone Company          104.89 .F. .T.
00002      2 Gas Company              4.14 .F. .T.
00003      3 Electricity            250.31 .F. .T.
00004      4 Grocery Store         1034.45 .F. .T.
00005     34 Me                     561.77 .T. .F.
00006      6 Bank, service charge     4.00 .T. .T.
00007      7 Doctor Doolittle       100.00 .T. .T.
00008      8 Pirates                101.01 .F. .T.
00009      9 Car Repair Man         500.01 .F. .T.
00010     10 Me                     561.01 .T. .F.
00011     11 Tuperware               50.02 .F. .T.
00012     12 Me                     561.77 .T. .F.
00013     13 Me                     750.03 .T. .F.

. USE MONTH
. DISP STRU
STRUCTURE FOR FILE:  MONTH.DBF
NUMBER OF RECORDS:   00003
DATE OF LAST UPDATE: 10/18/81
PRIMARY USE DATABASE
FLD       NAME       TYPE WIDTH   DEC
001      NUMBER       N    005
002      AMOUNT       N    010    002
003      HOME         L    001
** TOTAL **               00017

. LIST
00001        29      14.89 .T.
00002        16     764.09 .T.
00003        78      97.96 .T.

. APPEND FROM CHECKS FOR HOME
00006 RECORDS ADDED

. APPEND FROM CHECKS FOR OUTGOING
*** SYNTAX ERROR ***
                             ?
APPEND FROM CHECKS FOR OUTGOING
CORRECT AND RETRY(Y/N)? N
```

That last append was to show what would happen if the FOR field was not in the USE file.

BROWSE

BROWSE

The BROWSE command is one of the most powerful dBASE commands for data editing and viewing. The data from up to 19 records is displayed onto the screen (fewer if fields are greater than 80 characters). As many fields as will fit are put on each line. The screen should be considered as a window into a database. You can scroll backwards and forwards through the records and you can pan left and right through the fields of the database. Any data can be edited with the standard full-screen editing method (see section 8 for additional information).

This is a summary of the full-screen control keys that will work in BROWSE:

ctl-E,A	-	backs up to the previous data field;
ctl-X,F	-	advances to the next data field;
ctl-D	-	advances to the next character;
ctl-S	-	backs up to the last character;
ctl-G	-	deletes the character under the cursor;
RUBOUT	-	deletes the character before the cursor;
ctl-Q	-	exits without saving the changes;
ctl-W	-	exits and saves the changes (ctl-O for Superbrain);
ctl-B	-	pans the window left one field;
ctl-Z	-	pans the window right one field;
ctl-C	-	writes the current record and advances one record;
ctl-R	-	writes the current record and backs up one record;
ctl-U	-	switches (toggles) the current record between being marked for deletion and not being marked.

Example:

```
. BROWSE
```

CANCEL

CANCEL

Cancel a command file execution and return to the normal keyboard interpretive mode.

Example:

```
INPUT 'IS JOB DONE (Y/N)' TO X
IF X
  CANCEL
ENDIF
```

This is a fragment from a command file. The INPUT command asks for a yes/no answer. If the answer is yes ('Y', 'y', 'T', or 't') then the IF X line of the command file will be satisfied (since X will be logically .TRUE.) and the CANCEL command will be executed.

See Appendix A for more examples.

CHANGE

CHANGE [<scope>] FIELD <list> [FOR <exp>]

CHANGE is a command that allows the user to make a number of alterations to a database with minimum effort. All database fields that are referenced in the list are presented to the user in the order given by <list>. The user has the opportunity of entering new data, modifing the data or skipping to the next field. When the <list> has been exhausted, CHANGE will proceed to the next record as specified in the <scope>. The default scope is the current record.

A field can be deleted in its entirety by typing a control-Y (followed by a return) in response to the CHANGE? message. The CHANGE command can be aborted by typing an ESCAPE character.

Example:

```
 . USE CARDS
 . CHANGE FIELD DATE

 RECORD: 00001

 DATE: 08/19/81
 CHANGE? 81
 TO       82

 DATE: 08/19/82
 CHANGE? (cr)
```

CLEAR

CLEAR [GETS]

If the GETS (or GET) keyword is used then all of the GETs that are pending (i.e. a GET set up by the @ command) are cleared and the screen is left intact. This is opposed to the ERASE command which also clears pending GETs and also erases the screen.

If there is no GETS keyword, then this command resets dBASE II. All databases in USE are closed and un-used, all memory variables are released, and the PRIMARY work area is re-selected.

This command gives dBASE II a "clean slate". For instance: if a command file finished executing and left dBASE in the SECONDARY state, then executing a new command file that assumes that the PRIMARY state was selected, will cause unknown things to happen.

CLEAR should be used at the beginning of a command file to give the command file a known state.

Example:

```
 . CLEAR
```

CONTINUE

This command is used with the LOCATE command. LOCATE and CONTINUE may be separated by other commands, however there are limitations. See the LOCATE command for more information.

COPY

```
COPY TO <file> [<scope>] [FIELD <list>] [FOR <exp>]
        [SDF] [STRUCTURE] [DELIMITED [WITH <delimiter>]]
```

This command copies the database in USE to another file. The <file> may be in dBASE format or in the System Data Format (if the SDF option is specified).

If the STRUCTURE clause is specified, then only the structure of a dBASE file in USE is copied to the "TO" file.

If a list of fields is supplied following a FIELD clause, then only those data fields are copied TO the file. For the COPY STRUCTURE FIELD <list>, only the structure of the listed fields is copied TO the file. In either case, the new structure will be made up of only those fields specified by the FIELD clause. No FIELD clause specifies that all fields will be copied.

If the SDF clause is specified, then the file in USE is copied to another file without the structure. This new file will be in ASCII standard format. This allows the generation of files which can be input to processors other than dBASE. The STRUCTURE and SDF clauses are mutually exclusive.

If the DELIMITED keyword is also in the command, then the output file will have all of its character string type fields enclosed in quotes and the fields will be separated by commas. This is the converse of a delimited APPEND. By default, the DELIMITED type of COPY uses single quotes as delimiters to mark character string fields. The WITH sub-phrase of the DELIMITED phrase allows any character to be the delimiter. If a "," is used as the delimiter then the character fields will have trailing blanks trimmed, the numeric fields will have the leading blanks trimmed, and the character strings will not be enclosed in quotes. The APPEND command will only respond to single and double quotes.

If either the DELIMITED or SDF option is used then the output <file> name will default to a .TXT extension, otherwise the output file will default to a .DBF extension.

The "TO" file is created if it does not exist.

Examples:

```
. DISPLAY ALL OFF NAME,TELE:EXTSN
NEUMAN, ALFRED E.   1357
RODGERS, ROY        2468
CASSIDY, BUTCH      3344
CHANG, LEE          6743
POST, WILEY         1011
LANCASTER, WILLIAM J 6623
NORRIS, R. "BOB"    8093

. DISPLAY STRUCTURE
STRUCTURE FOR FILE:    EXAMPLE
NUMBER OF RECORDS:     00007
DATE OF LAST UPDATE:   00/00/00
PRIMARY USE DATABASE
FLD      NAME        TYPE WIDTH    DEC
001      NAME         C    020
002      TELE:EXTSN   C    005
003      MAIL:STOP    C    010
** TOTAL **               00036

. COPY TO DUPE
00007 RECORDS COPIED

. COPY TO DUPE2 FOR TELE:EXTSN<'8000'
00006 RECORDS COPIED

. USE DUPE2

. DISPLAY ALL
00001  NEUMAN, ALFRED E.     1357  123/456
00002  RODGERS, ROY          2468  180/103
00003  CASSIDY, BUTCH        3344  264/401
00004  CHANG, LEE            6743  190/901
00005  POST, WILEY           1011  84/13B
00006  LANCASTER, WILLIAM J 6623  170/430

. USE EXAMPLE

. COPY FIELD NAME,TELE:EXTSN TO DUPE3
00007 RECORDS COPIED
.
. USE DUPE3
```

```
STRUCTURE FOR FILE:   DUPE3
NUMBER OF RECORDS:    00007
DATE OF LAST UPDATE:  00/00/00
PRIMARY USE DATABASE
FLD       NAME      TYPE WIDTH   DEC
001      NAME         C    020
002      TELE:EXTSN   C    005
** TOTAL **              00036

. DISPLAY ALL
00001  NEUMAN, ALFRED E.      1357
00002  RODGERS, ROY           2468
00003  CASSIDY, BUTCH         3344
00004  CHANG, LEE             6743
00005  POST, WILEY            1011
00006  LANCASTER, WILLIAM J   6623
00007  NORRIS, R. "BOB".      8093
. USE EXAMPLE

. COPY NEXT 4 TO DUPE5
00004 RECORDS COPIED

. USE DUPE5

. DISPLAY ALL
00001  NEUMAN, ALFRED E.      1357  123/456
00002  RODGERS, ROY           2468  180/103
00003  CASSIDY, BUTCH         3344  264/401
00004  CHANG, LEE             6743  190/901
```

(The delimited COPY)

```
. USE ORDERS

. DISP STRUCTURE
STRUCTURE FOR FILE:   ORDERS.DBF
NUMBER OF RECORDS:    00012
DATE OF LAST UPDATE:  07/01/80
PRIMARY USE DATABASE
FLD       NAME      TYPE WIDTH   DEC
001      CUSTOMER     C    020
002      PART:NO      C    005
003      AMOUNT       N    005
** TOTAL **              00031
```

```
. LIST
00001  SWARTZ, JOE              31415    13
00002  SWARTZ, JOE              76767    13
00003  HARRIS, ARNOLD           11528    44
00004  ADAMS, JEAN              89793    12
00005  MACK, JAY                31415     3
00006  TERRY, HANS              76767     5
00007  JUAN, DON                21828     5
00008  SALT, CLARA              70296     9
00009  BARNETT, WALT            31415     6
00010  NICHOLS, BILL            76767    17
00011  MURRAY, CAROL            89793     4
00012  WARD, CHARLES A.         92653    15

. COPY TO DELIM.DAT DELIMITED
00012 RECORDS COPIED

'SWARTZ, JOE           ','31415',   13
'SWARTZ, JOE           ','76767',   13
'HARRIS, ARNOLD        ','11528',   44
'ADAMS, JEAN           ','89793',   12
'MACK, JAY             ','31415',    3
'TERRY, HANS           ','76767',    5
'JUAN, DON             ','21828',    5
'SALT, CLARA           ','70296',    9
'BARNETT, WALT         ','31415',    6
'NICHOLS, BILL         ','76767',   17
'MURRAY, CAROL         ','89793',    4
'WARD, CHARLES A.      ','92653',   15
```

COUNT

COUNT [<scope>] [FOR <exp>] [TO <memvar>]

Count the number of records in the USE file. If the FOR clause is invoked, then only the number of records which satisfy the expression are counted. If the TO clause is included, the integer count is places into a memory variable. The memory variable will be created if it did not exist prior to this command.

dBASE responds with the message:

```
 COUNT = xxxxx
```

Examples:

```
 . USE INVNTRY

 . DISPLAY STRUCTURE
 STRUCTURE FOR FILE:    INVNTRY
 NUMBER OF RECORDS:     00010
 DATE OF LAST UPDATE:   10/23/78
 PRIMARY USE DATABASE
 FLD       NAME      TYPE WIDTH   DEC
 001      ITEM:NO     N    006
 002      CLASS:NO    N    003
 003      VENDOR:NO   N    005
 004      DESCR       C    013
 005      UNIT:COST   N    007    002
 006      LOCATION    C    005
 007      ON:HAND     N    004
 008      SOLD        N    004
 009      PRICE       N    007    002
 ** TOTAL **               00055

 . DISPLAY ALL
 00001  136928  13  1673 ADJ. WRENCH      7.13 18     9   0    9.98
 00002  221679   9  1673 SM. HAND SAW     5.17 17     4   1    7.98
 00003  234561   0    96 PLASTIC ROD      2.18 27   112  53    4.75
 00004  556178   2   873 ADJ. PULLEY     22.19 117    3   0   28.50
 00005  723756  73    27 ELECT.BOX       19.56 354    6   1   29.66
 00006  745336  13    27 FUSE BLOCK      12.65 63     7   2   15.95
 00007  812763   2  1673 GLOBE            5.88 112    5   2    7.49
 00008  876512   2   873 WIRE MESH        3.18 45     7   3    4.25
 00009  915332   2  1673 FILE             1.32 97     7   3    1.98
 00010  973328   0    27 CAN COVER        0.73 21    17   5    0.99

 . COUNT
 COUNT = 00010


 . COUNT FOR ITEM:NO>500000
 COUNT = 00007
```

```
. COUNT FOR 'ADJ'$DESCR
COUNT = 00002

. GOTO TOP

. COUNT FOR PRICE<10 NEXT 6
COUNT = 00003

. GOTO TOP

. COUNT NEXT 6 FOR PRICE<10
COUNT = 00003

. USE B:SHOPLIST

. LIST
00001  BEANS                  5      0.75
00002  BREAD LOAVES           2      0.97
00003  T-BONE                 4      3.94
00004  PAPER PLATES           1      0.86
00005  PLASTIC FORKS          5      0.42
00006  LETTUCE                2      0.53
00007  BLEU CHEESE            1      1.96
00008  MILK                   2      1.30
00009  CHARCOAL               2      0.75

. DISPLAY STRUCTURE
STRUCTURE FOR FILE:    B:SHOPLIST.DBF
NUMBER OF RECORDS:     00009
DATE OF LAST UPDATE:   12/10/76
PRIMARY USE DATABASE
FLD       NAME      TYPE WIDTH   DEC
001      ITEM         C   020
002      NO           N   005
003      COST         N   010    002
** TOTAL **              00036

. COUNT TO XX FOR COST>1
COUNT = 00003

. ? XX
 3
```

CREATE

```
------
```

CREATE [<filename>]

A new dBASE structured file is CREATEd. The user provides the structure, field names, and file name for the database file.

If not supplied in the command, the user is first prompted for the <filename> to be used by the message:

FILENAME:

The user enters a valid filename with the following added restriction: the filename may contain no special characters other than those normally used by CP/M for special purposes (such as B: to denote disk drive "B").

If the file existed before the create command was given, dBASE asks the user:

DESTROY EXISTING FILE? To which the user must reply Y or N as the case may be.

If the file is new to the system or if the user answered Y to the destroy question, dBASE is now ready to accept the structure of the data base from the user. The following message is displayed:

```
ENTER RECORD STRUCTURE AS FOLLOWS:
 FIELD    NAME,TYPE,WIDTH,DECIMAL PLACES
 001
```

The user now enters field names and associated structure information. A field name is a character string up to 10 characters long which consists of alphabetic letters, numeric digits, and colons. Field names must begin with an alphabetic character. Fields may be any of three types: character string, numeric, or logical. The type field is specified by one character, as:

C - character string
N - numeric
L - logical

The width refers to the length of the field, for instance, a character string may be 20 characters long i.e. it's width is 20. Numeric data may be either integer or decimal. The width of integers is the maximum number of digits that they may be expected to contain. For decimal numbers, two widths are required; the first is the maximum number of digits that the decimal number is expected to contain (including the decimal point), the second width is the number of digits which are to by allowed on the right side of the decimal point. Logical data may only be of length 1.

Examples:

```
. CREATE
FILENAME:EXAMPLE
ENTER RECORD STRUCTURE AS FOLLOWS:
 FIELD     NAME,TYPE,WIDTH,DECIMAL PLACES
 001      NAME,C,20
 002      TELE:EXTSN,C,5
 003      MAIL:STOP,C,10
 004      (cr)
INPUT NOW?Y

RECORD 00001

NAME:         NEUMAN, ALFRED E.
TELE:EXTSN:  1357
MAIL:STOP:   123/456

RECORD 00002

NAME:         RODGERS, ROY
TELE:EXTSN:  2468
MAIL:STOP:   180/103

RECORD 00003

NAME:         CASSIDY, BUTCH
TELE:EXTSN:  3344
MAIL:STOP:   264/401

RECORD 00004

NAME:         CHANG, LEE
TELE:EXTSN:  6743
MAIL:STOP:   190/901
```

```
RECORD 00005

NAME:         POST, WILEY
TELE:EXTSN:  1011
MAIL:STOP:   84/13B

RECORD 00006

NAME:         (cr)

. DISPLAY STRUCTURE
NO FILE IN USE, FILENAME:  EXAMPLE
STRUCTURE FOR FILE:    EXAMPLE
NUMBER OF RECORDS:     00005
DATE OF LAST UPDATE:   00/00/00
PRIMARY USE DATABASE
FLD      NAME        TYPE WIDTH   DEC
001      NAME          C    020
002      TELE:EXTSN    C    005
003      MAIL:STOP     C    010
** TOTAL **               00036

. DISPLAY ALL
00001  NEUMAN, ALFRED E.      1357  123/456
00002  RODGERS, ROY           2468  180/103
00003  CASSIDY, BUTCH         3344  264/401
00004  CHANG, LEE             6743  190/901
00005  POST, WILEY            1011  84/13B
```

DELETE

DELETE [<scope>] [FOR <exp>]
DELETE FILE <filename>

All records which are within <scope> (and which satisfy the FOR expression if present) are marked for deletion. The default scope is the current record only. Records are not physically deleted until a PACK operation, however records marked for deletion will not be copied, appended, or sorted. The RECALL operation may be used to revive records marked as deleted. Records which are marked for deletion can be displayed. The mark of deletion appears as an asterisk between the record number and the first field.

In the second form, the file named <filename> will be removed from the disk drive where it resides (if possible) and the space it was occupying will be released to the operating system for reassignment. If, however, the <filename> is currently in use, the file will not be deleted.

Examples:

```
. LIST
00001  136928  13  1673 ADJ. WRENCH       7.13 189     9   0    9.98
00002  221679   9  1673 SM. HAND SAW      5.17 173     4   1    7.98
00003  234561   0    96 PLASTIC ROD       2.18 27    112  53    4.75
00004  556178   2   873 ADJ. PULLEY      22.19 117     3   0   28.50
00005  723756  73    27 ELECT.BOX        19.56 354     6   1   29.66
00006  745336  13    27 FUSE BLOCK       12.65 63      7   2   15.95
00007  812763   2  1673 GLOBE             5.88 112     5   2    7.49
00008  876512   2   873 WIRE MESH         3.18 45      7   3    4.25
00009  915332   2  1673 FILE              1.32 97      7   3    1.98

. DELETE RECORD 2
00001 DELETION(S)

. 5

. DELETE NEXT 3
00003 DELETION(S)
```

```
. LIST
00001  136928  13  1673 ADJ. WRENCH      7.13 189      9   0   9.98
00002 *221679   9  1673 SM. HAND SAW     5.17 173      4   1   7.98
00003  234561   0    96 PLASTIC ROD      2.18 27     112  53   4.75
00004  556178   2   873 ADJ. PULLEY     22.19 117      3   0  28.50
00005 *723756  73    27 ELECT.BOX       19.56 354      6   1  29.66
00006 *745336  13    27 FUSE BLOCK      12.65 63       7   2  15.95
00007 *812763   2  1673 GLOBE            5.88 112      5   2   7.49
00008  876512   2   873 WIRE MESH        3.18 45       7   3   4.25
00009  915332   2  1673 FILE             1.32 97       7   3   1.98

. RECALL ALL
00004 RECALL(S)

. LIST
00001  136928  13  1673 ADJ. WRENCH      7.13 189      9   0   9.98
00002  221679   9  1673 SM. HAND SAW     5.17 173      4   1   7.98
00003  234561   0    96 PLASTIC ROD      2.18 27     112  53   4.75
00004  556178   2   873 ADJ. PULLEY     22.19 117      3   0  28.50
00005  723756  73    27 ELECT.BOX       19.56 354      6   1  29.66
00006  745336  13    27 FUSE BLOCK      12.65 63       7   2  15.95
00007  812763   2  1673 GLOBE            5.88 112      5   2   7.49
00008  876512   2   873 WIRE MESH        3.18 45       7   3   4.25
00009  915332   2  1673 FILE             1.32 97       7   3   1.98

. DISP FILES ON B
DATABASE FILES    # RCDS     LAST UPDATE
SHOPLIST          00007      06/06/76
SHOPSAVE          00007      06/05/76

. DELETE FILE B:SHOPSAVE
FILE DELETED

. DISPLAY FILES ON B
DATABASE FILES    # RCDS     LAST UPDATE
SHOPLIST          00007      06/06/76
```

DISPLAY
----

a. DISPLAY [<scope>] [FOR <exp>] [<exp list>] [OFF]
b. DISPLAY STRUCTURE
c. DISPLAY MEMORY
d. DISPLAY FILES [ON <disk drive>] [LIKE <skeleton>]

Display is the foundation of dBASE. The end goal of all database operation is to display the data in the database (or cross sections and abstractions of the data) upon demand. DISPLAY satisfies that goal by allowing a wide variety of forms that select the wanted data.

In case a. all or part of the database in USE is displayed. If <scope> is not specified and the FOR <exp> is not in the command, only the current record can contribute information for display. If <scope> is not specified and there is a FOR <exp>, then all records in the database may contribute to the display. All fields are displayed unless the <exp list> clause is specified. Valid expressions may consist of data fields, memory variables, or any valid literal number, character or logical. The current record number is prefixed to each line displayed unless the OFF option is selected. If the FOR clause is specified, then only those records that satisfy the FOR's conditional expression can contribute information for display.

After groups of 15 records have been displayed, DISPLAY waits for any keystroke to continue. This allows the user to "page" through a long display. The LIST command is identical to the DISPLAY command except that LIST does not wait after record groups and it's default scope is ALL records. An ESCape character terminates the DISPLAY or LIST commands.

In case b. only the structure of the database in USE is displayed.

In case c. all currently defined memory variables are displayed as memory variable name and associated value.

Case d. is a way to display .DBF files that are residing on the default unit (or on <disk drive>) along with some of the database's statistics. The LIKE phrase allows other types of files to be displayed. The <skeleton> is usually of the form *.type, where type is TXT, FRM, MEM, or any other three letter string. These files are displayed just as in the CP/M DIR command.

Examples:

```
. USE B:INVENTRY

. DISPLAY STRUCTURE
STRUCTURE FOR FILE:  B:INVENTRY.DBF
NUMBER OF RECORDS:  00008
DATE OF LAST UPDATE: 00/00/00
PRIMARY USE DATABASE
FLD      NAME      TYPE WIDTH   DEC
001     ITEM        C    020
002     COST        N    010    002
003     PART:NO     C    005
004     ON:HAND     N    005
** TOTAL                00041 BYTES      (note: total includes
                                          1 overhead byte)

. DISPLAY ALL ITEM, PART:NO, COST*ON:HAND ,$(PART:NO,1,2) FOR ;
 COST > 100 .AND. ON:HAND > 2 OFF
TANK, SHERMAN        89793          404997.00  89
TROMBONES            76767           15076.12  76
RINGS, GOLDEN        70296            1000.00  70

. DISPLAY MEMORY
CLIENT:NAM (C)  DANGLEMEYER, PRENTICE
BUDGET     (N)   123456.70
EF:STATUS  (L)  .T.
** TOTAL **     03 VARIABLES USED  00027 BYTES USED

. DISPLAY FILES ON B: LIKE *.FRM
TEST      FRM      ADMIN     FRM       ORDERS    FRM

. DISPLAY FILES
DATABASE FILES  #RCDS   LAST UPDATE
TEST      DBF   00077   00/00/00
ADRECS    DBF   00073   09/23/81
HISTSTR   DBF   00000   06/29/81
TMPADMIN DBF
NOT A dBASE II DATABASE
```

The last .DBF file in the list above is the file that is not the dBASE database.

Only representative examples of DISPLAY are given here, refer to other commands for other examples.

DO
--

```
a. DO <file>
b. DO WHILE <exp>
   <statements>
   ENDDO
c. DO CASE
     CASE <exp>
        <statements>
     CASE <exp>
        <statements>
        .
        .
        .
     [OTHERWISE]
        <statements>
   ENDCASE
```

In case a, <file> is opened and read. The file in this case is known as a COMMAND FILE. It consists entirely of dBASE commands. The input is interpreted and executed as keyboard commands are. DO's can be stacked up to 16 deep (i.e. command files can contain DO commands which invoke other command files). Control is released by a command file with an end-of-file or by the RETURN command. If the current command file was called by a command file, control will be given back to the higher level command file. If, during the execution of a command file, a CANCEL command is encountered; all command files are closed and the keyboard is made the source for future commands.

In case b, if the <exp> evaluates as a logical TRUE, the statements following the DO are executed until an ENDDO statement is encountered. If the <exp> evaluates to a logical FALSE, control is transferred to the statement following the ENDDO statement.

Note: <statements> refers to entire statements. The DO WHILE statement ends with an ENDDO. Statements must nest properly; if there is an IF "inside" a DO WHILE, then an ENDDO may not occur before the ENDIF. See section 9.2 Rule 8 for more information.

Examples:

```
DO ACCNTPAY

DO WHILE .NOT.EOF
  DISPLAY NAME
  .
  .
  .
  SKIP
ENDDO
```

CASE is an extension of the DO command and takes the form shown above. There is no limit to the number of CASE phrases that a DO CASE may contain. The OTHERWISE phrase is optional.

DO CASE is a structured procedure. The individual CASEs in the construct could be viewed as the exceptions to the rule that defines the OTHERWISE. If some condition needs some special processing then the condition would be a CASE and all other conditions would be the OTHERWISE. OTHERWISE may also be viewed as the default condition. See the first example below.

How dBASE handles the DO CASE construct may best be explained as a series of IFs. That is, dBASE will execute the DO CASE as if it were a list of IF-ENDIFs.

```
DO CASE                              IF ITEM='ORANGES'
   CASE ITEM='ORANGES'                  any statements
     any statements                  ELSE
   CASE ITEM='APPLES'          =        IF ITEM='APPLES'
     any statements                        any statements
   OTHERWISE                            ELSE
     any statements                        any statements
   ENDCASE                              ENDIF
                                     ENDIF
```

Thus, dBASE will examine the <exp>s in the individual CASEs and the first one that is true will have the statements after it executed. When dBASE reaches the next phrase beginning with a "CASE" it will exit to the ENDCASE. This means that if more than one CASE is true, only the first one will be executed.

If the OTHERWISE clause is present and none of the CASEs are true, then the <statements> in the OTHERWISE clause will be executed. If there is no OTHERWISE clause and none of the CASEs are true, then the DO CASE will be exited with none of the <statements> executed at all.

Any statements that are placed between the "DO CASE" and the first "CASE" will not be executed.

Examples:

```
DO CASE
   CASE ITEM = "BROWN"
        <statements> that process BROWN
   CASE ITEM = "JONES"
        <statements> that process JONES
   CASE ITEM = "SMITH"
        <statements> that process SMITH
   OTHERWISE
        <statements> that process all the other names
ENDCASE
```

In the case above all the expressions were for the same field name. This is not necessary. An <exp> may contain anything and the series of CASEs need not have a tight relationship.

```
DO CASE
   CASE TODAY = "MONDAY"
        <statements> for MONDAY
   CASE WEATHER = "RAIN"
        <statements> for RAIN
   CASE CITY = "LOS ANGELES"
        <statements> for LOS ANGELES
ENDCASE
```

Of course, if it is a rainy Monday in Los Angeles only the CASE for MONDAY will be executed.

CASEs need not be all character strings as in these two examples. Any expression will work.

```
DO CASE
   CASE 3 = 2 + 1
        <statements> for addition
   CASE .NOT. A
        <statements> for boolean logic
   CASE "A"$"ABCDEF"
        <statements> for string logic
   OTHERWISE
        <statements>
ENDCASE
```

ENDCASE is the statement used to terminate a DO CASE structure. When a case or OTHERWISE has finished processing, control is resumed at the line following the ENDCASE.

EDIT

EDIT [n]

The EDIT command allows the user to selectively change the contents of the data fields in a database. Edit's usage and action varies, depending on whether on not dBASE is in the full-screen mode (see the SET SCREEN command).

When dBASE is in the full-screen mode, editing can be done by either "EDIT" or "EDIT n" (n represents the record to be edited). If n is not present then dBASE will ask for the coordinates of the record to be edited. This is similar to the non-full-screen mode, however, full-screen capabilities will still used after the record number is supplied. See section 8, full-screen operations, for a description of control keys and cursor movement.

When the edit command is used in the non-full-screen mode, dBASE responds with:

COORD:

The user then enters the coordinates of the data field to be changed and (optionally) the new value. The coordinates of the data field are: the record number, and the field number (or the field name). If a new value is supplied, dBASE will replace the contents of the specified field with the new value. If a new value is not supplied, dBASE displays the current value of the data field and prompts the user for changes. If no changes are desired, a carriage return will cause dBASE not to alter the contents of the field. Whether changes are made or not, dBASE will prompt the user for the next pair of coordinates with another "COORD:" message.

After the first set of coordinates have been entered, the user may omit either of the coordinate values and dBASE will use the previous value of that coordinate. The EDIT mode is exited by entering a carriage return as the response to the COORD request.

The entire data field can be erased by entering a control-Y, RETURN whenever the CHANGE? message is displayed. This permits a field to be completely reentered if desired. The editing of a data field can be aborted by entering a CTL-Q character. This discards any editing done and restores the data field to its original contents.

If an INDEXed file is being EDITed and the index clause was USEd, then dBASE will adjust the index if the key field is altered. If more than one index file is associated with the database, then the un-USEd files will be unaffected by the edit.

Examples:

```
   USE SHOPLIST
```

```
. DISPLAY STRUCTURE
STRUCTURE FOR FILE:     SHOPLIST
NUMBER OF RECORDS:      00006
DATE OF LAST UPDATE:    07/03/76
PRIMARY USE DATABASE
FLD       NAME        TYPE WIDTH   DEC
001     ITEM            C   020
002     NO              N   005
003     COST            N   010    002
** TOTAL **                00036

. LIST
00001  BEANS #303 CAN              5          0.69
00002  BREAD                       2          0.89
00003  T-BONE STEAKS               4          3.59
00004  LETTUCE                     1          0.49
00005  MILK (1 GAL BOTTLES)        2          1.19
00006  CHARCOAL                    1          0.69

. EDIT
COORD: 5,ITEM,MILK (1/2 GAL)

COORD: 2,1

ITEM: BREAD

CHANGE? D
TO      D LOAVES

ITEM: BREAD LOAVES
CHANGE? (cr)
COORD: 6,1

ITEM: CHARCOAL
CHANGE? AL
TO      AL, 5# BAGS

ITEM: CHARCOAL, 5# BAGS
CHANGE? (cr)
COORD: ,2

NO:      1
TO: 2
COORD: 4

NO:      1
TO: 2
COORD: (cr)
. LIST
00001  BEANS #303 CAN              5          0.69
00002  BREAD LOAVES                2          0.89
00003  T-BONE STEAKS               4          3.59
00004  LETTUCE                     2          0.49
00005  MILK (1/2 GAL)              2          1.19
```

```
00006  CHARCOAL, 5# BAGS          2        0.69
```

(The following portion of a command file would also allow one to edit a database on a selective basis. The "&" is vital to making these commands work; it will change the string accepted by the ACCEPT into numbers that EDIT will recognize.)

```
STORE '1' TO X
DO WHILE X <> '0'
  ACCEPT "Enter Record Number" TO X
  EDIT &X
ENDDO
```

EJECT

EJECT

This command causes the printer to do a form feed (eject the page) if either PRINT is SET ON or FORMAT is SET TO PRINT. When using the @ command to do direct page formatting, the EJECT command also zeros the line and column registers.

Example:

```
EJECT
```

ENDDO

The statement used to terminate a DO WHILE loop. When encountered, control is transferred back to the DO statement for re-assessment of the logical value of the <exp>.

See the DO command.

See Appendix A for examples.

ERASE

ERASE

This command clears the screen and places the cursor (if any) in the upper left corner of the screen. When using the @ command with the SET SCREEN ON in effect, ERASE clears memory of prior @ command gets and pictures.

Example:

```
. ERASE
```

FIND

FIND <char string> or '<char string>'

This command causes dBASE to FIND the first record in an indexed database (in USE) whose key is the same as <char string>. FIND allows very rapid location of records within an indexed database. A typical FIND time is two seconds on a floppy diskette system.

FIND operates only on databases that have previously been indexed (see the INDEX command description). If the INDEX command used a character string expression as the key, then FIND will operate when it is given only the first few characters of the key. The found record will be the first one whose key has the same order and number of characters as the <char string>. Fcr example: a record whose key is 'SMITH, JOHN' could be found by the statement 'FIND SMI' provided that there are no other keys starting with 'SMI' proceeding SMITH, JOHN in the index. FIND will always find only the first record whose key is the same as <char string>. Even if the record pointer is moved down further in the file, a subsequent FIND on the same key will find the FIRST record.

If the index was created with a numeric key, then the found record will be the first record whose key is arithmetically equal to the object of the FIND.

Note: that for indexes keyed on both characters and numbers, the FIND object is a character string with or without quote delimiters. Quote marks only become necessary for character strings if the original key had leading blanks. In that case, the exact number of leading blanks should be inside the quotes.

If a memory variable is desired as a FIND object, it must be placed after the FIND command by means of an &-macro replacement, e.g. FIND &NAME where NAME is a character string memory variable. Numeric memory variables must first be converted to a string by means of the STR function before they can be "macro-ized". See section 5 for a discussion on macros.

Once a record in a database has been located by means of the FIND command, it can be processed just as any other database record. That is, it can be interrogated, altered, used in calculations, etc. dBASE commands that cause movement of the database (e.g. LIST, REPORT, COPY, etc.) will process the found record first and proceed to the next record in sequence, based upon the key.

If no record exists whose key is identical to the <char string> then the message: "NO FIND" will be displayed on the screen and the record number function "#" will give the value of zero.

If a second record with the same key is wanted, then a SKIP or a LOCATE FOR <exp> should be used. The SKIP will not know when there is no longer a match, the LOCATE (as long as the key was

used in the expression) will be able to find additional matches.

SET EXACT ON will cause FIND to get a 'hit' only if there is a character for character match for the ENTIRE key (except for trailing blanks).

Examples:

```
. USE SHOPLIST INDEX SHOPINDX

. LIST
00001  Beans            5    0.75
00007  Bleu cheese      1    1.96
00002  Bread loaves     2    1.06
00009  Charcoal         2    0.75
00006  Lettuce          2    0.53
00008  Milk             2    1.30
00004  Paper plates     1    0.94
00005  Plastic forks    5    0.42
00003  T-Bone steak     4    4.33

. FIND Bread

. DISPLAY
00002  Bread loaves     2    1.06

. DISPLAY NEXT 3
00002  Bread loaves     2    1.06
00009  Charcoal         2    0.75
00006  Lettuce          2    0.53

. FIND P

. DISPLAY
00004  Paper plates     1    0.94

. FIND Plas

. DISPLAY
00005  Plastic forks    5    0.42

. FIND P

. DISPLAY
00004  Paper plates     1    0.94
```

FIND will work in a multiple indexed file if the two keys are placed within quotes.

```
. list

00001  Flying High             Bird, I. M.      IMB001 02/29/04
00005  Nesting Procedures      Bird, I. M.      IMB002 09/25/06
00002  Diving                  Fish, U. R.      URF001 12/30/23
00008  Nursing                 Knight and Gale  KG001  08/04/44
00010  Vacationing in Europe   Knight and Gale  KG002  06/24/42
00004  101 Ways to Tie a Knot  Lynch, I.        IL001  04/01/00
00003  How to Survive a Crash  Lynch, M.        ML001  01/01/30
00007  Even Primes             Sladek, L        LS001  12/01/73
00009  Even More Primes        Sladek, L        LS002  04/24/73
00006  Thinking Big            Tim, Tiny        TT001  05/07/42

. find "Bird, I. M.    IMB002"

. disp
00005  Nesting Procedures      Bird, I. M.      IMB002 09/25/06

. find "Lynch, M."

. disp
00003  How to Survive a Crash  Lynch, M.        ML001  01/01/30

. find "Sladek, L      LS002"

. disp
00009  Even More Primes        Sladek, L        LS002  04/24/73
```

GO or
GOTO
----

a. GOTO RECORD <n>
b. GOTO TOP
c. GOTO BOTTOM
d. <n>
e. GOTO <memvar>

This command is used to reposition the record pointer of the database.

In either case a or d, the current-record pointer is set to record number <n>. Case d is a short-hand method for case a.

In cases b and c, the file in USE is rewound/unwound (TOP/BOTTOM) and the first/last record in the file is pointed to by the current-record pointer. When the file in USE has been INDEXed, then first/last record is not necessarily the first/last physical record in the database but rather is first/last according to the key used to index the database.

Case e can be used to position to a record number contained in a memory variable.

Examples:

```
. USE SHOPLIST

. GOTO RECORD 6
  6

. DISPLAY
00006  LETTUCE            2        0.53

. GOTO TOP

. DISPLAY
00001  BEANS              5        0.75

. GOTO BOTTOM

. DISPLAY
00009  CHARCOAL           2        0.75
```

```
. LIST
00001  BEANS                5       0.75
00002  BREAD LOAVES         2       0.97
00003  T-BONE               4       3.94
00004  PAPER PLATES         1       0.86
00005  PLASTIC FORKS        5       0.42
00006  LETTUCE              2       0.53
00007  BLEU CHEESE          1       1.96
00008  MILK                 2       1.30
00009  CHARCOAL             2       0.75

. STORE 4 TO RECORDNO
4

. GOTO RECORDNO

. DISP
00004  PAPER PLATES                 0.86
```

IF

```
IF <exp>
   <commands>
[ELSE
   <commands>]
ENDIF
```

The IF command allows conditional execution of other commands. This command is used in command files. When the <exp>ression evaluates to TRUE, the commands following the IF are executed. When the expression evaluates to FALSE, the commands following the ELSE are executed. If no ELSE is specified, all commands are skipped until an ENDIF is encountered. IF commands may be nested to any level.

Note: <commands> refers to whole command statements. The IF command begins with IF and ends with ENDIF. Statements must nest properly, an IF with a DO WHILE in the true (or false) path must not end before the DO WHILE. See section 9.8 Rule 8 for more information.

Examples:

```
IF STATUS='MARRIED'
   DO MCOST
ELSE
   DO SCOST
ENDIF

IF X=1
  STORE CITY+STATE TO LOCATION
ENDIF
```

See Appendix A for further examples.

INDEX

INDEX ON <expression> TO <index file name>

The INDEX command causes the current file in USE to be indexed on the <expression>. <expression> is known as the "key". This means that a file will be constructed by dBASE (the <index file>) that contains pointers to the records in the USE file. The index file is made in such a way that the USE database appears to be sorted on the key for subsequent operations. The file in use is not physically changed. Sorting will be in an ascending order. A descending sort may be done on an expression that is a numeric. See below for an example.

Indexing allows very rapid location of database records by specifing all or part of the key by means of the FIND command. (See FIND). A database need not be indexed unless the application being worked would be enhanced by it. An indexed database can be used later with or without the indexing feature.

Many times, the INDEX command need only be done once for any given file. For instance, the APPEND command will automatically adjust the index file when new records are added.

If an indexed database is reUSEd (in a later dBASE run or later in the same run that did the original INDEX operation), then a special form of the USE command must be used (i.e. USE <database filename> INDEX <index filename>).

Any number of index files may be constructed for any database, however, only the USEd index files will be automatically updated by the APPEND, EDIT, REPLACE, READ or BROWSE commands.

An indexed file can be packed with the PACK command and the database, as well as the index file, will be properly adjusted. However if more that one index file is associated with the PACKed database, then that database must reINDEXed on those keys.

WARNING: The TRIM function must NOT be used as part of an index key. Also, if the $ or STR functions are used as part or all of a key, they must have literal numbers (not variables or expressions) as their length parameters (e.g. INDEX ON $(NAME,N,5)+STR(AMOUNT,5) TO NDXFILE instead of INDEX ON $(NAME,N,N+5)+STR(AMOUNT,SIZEVAR) TO NDXFILE).

Examples:

```
. USE SHOPLIST

. LIST
00001  Beans                    5      0.75
00002  Bread loaves             2      1.06
00003  T-Bone steak             4      4.33
00004  Paper plates             1      0.94
00005  Plastic forks            5      0.42
00006  Lettuce                  2      0.53
00007  Bleu cheese              1      1.96
00008  Milk                     2      1.30
00009  Charcoal                 2      0.7u

. DISPLAY STRUCTURE
STRUCTURE FOR FILE:    SHOPLIST.DBF
NUMBER OF RECORDS:     00009
DATE OF LAST UPDATE:   07/03/76
PRIMARY USE DATABASE
FLD       NAME      TYPE WIDTH   DEC
001       ITEM       C     020
002       NO         N     005
003       COST       N     010    002
** TOTAL **              00036

. NOTE CREATE INDEX FILE SHOPINDX

. INDEX ON ITEM TO SHOPINDX

. NOTE NOW LIST IN INDEX ORDER

. LIST
00001  Beans                    5      0.75
00007  Bleu cheese              1      1.96
00002  Bread loaves             2      1.06
00009  Charcoal                 2      0.75
00006  Lettuce                  2      0.53
00008  Milk                     2      1.30
00004  Paper plates             1      0.94
00005  Plastic forks            5      0.42
00003  T-Bone steak             4      4.33

. NOTE INDEXING ALLOWS FIND COMMAND

. FIND Milk

. DISPLAY
00008  Milk                     2.     1.30

. FIND Be

. DISPLAY
00001  Beans                    5      0.75
. SKIP
```

```
RECORD: 00007

. DISPLAY
00007  Bleu cheese                1          1.96

. SKIP -1
RECORD: 00001

. DISPLAY
00001  Beans                      5          0.75

. NOTE REGULAR USE COMMAND DOES NOT INCLUDE INDEX FILE

. USE SHOPLIST

. LIST
00001  Beans                      5          0.75
00002  Bread loaves               2          1.06
00003  T-Bone steak               4          4.33
00004  Paper plates               1          0.94
00005  Plastic forks              5          0.42
00006  Lettuce                    2          0.53
00007  Bleu cheese                1          1.96
00008  Milk                       2          1.30
00009  Charcoal                   2          0.75

. NOTE ALTERNATE FORM OF USE COMMAND RECALLS INDEX FILE

. USE SHOPLIST INDEX SHOPINDX

. LIST
00001  Beans                      5          0.75
00007  Bleu cheese                1          1.96
00002  Bread loaves               2          1.06
00009  Charcoal                   2          0.75
00006  Lettuce                    2          0.53
00008  Milk                       2          1.30
00004  Paper plates               1          0.94
00005  Plastic forks              5          0.42
00003  T-Bone steak               4          4.33
```

```
. USE BOOKS
. DISP STRU

STRUCTURE FOR FILE:   BOOKS.DBF
NUMBER OF RECORDS:    00010
DATE OF LAST UPDATE:  10/18/81
PRIMARY USE DATABASE
FLD       NAME      TYPE WIDTH   DEC
001       TITLE      C    025
002       AUTHOR     C    015
003       CAT:NUM    C    006
004       ARR:DTE    C    008
** TOTAL **             00055

. INDEX ON AUTHOR + CAT:NUM TO BOOKS
00010 RECORDS INDEXED

. LIST
00001  Flying High                Bird, I. M.        IMB001 02/29/04
00005  Nesting Procedures         Bird, I. M.        IMB002 09/25/06
00002  Diving                     Fish, U. R.        URF001 12/30/23
00008  Nursing                    Knight and Gale KG001  08/04/44
00010  Vacationing in Europe      Knight and Gale KG002  06/24/42
00004  101 Ways to Tie a Knot     Lynch, I.          IL001  04/01/00
00003  How to Survive a Crash     Lynch, M.          ML001  01/01/30
00007  Even Primes                Sladek, L          LS001  12/01/73
00009  Even More Primes           Sladek, L          LS002  04/24/73
00006  Thinking Big               Tim, Tiny          TT001  05/07/42
```

INPUT

INPUT ["<cstring>"] TO <memvar>

This construct permits the entry of expression values into memory variables, and can be used within command files as a means for the user to enter data at the command file's bidding. <memvar> is created, if necessary, and the expression is stored into <memvar>. If <cstring> is present, it is displayed on the screen as a prompt message before the input is accepted.

The type of the <memvar> is determined from the type of data that is entered. If a delimited character string is entered, the <memvar> will be of type character. If a numeric expression is entered, <memvar> will be of type numeric. If a T or Y (for True or Yes) is entered, <memvar> will be a logical variable with the value TRUE; if an F or N (for False or No) is entered, <memvar> will be a logical variable with the value FALSE. The function TYPE may be used to explicitly determine the type of the entry.

Either single or double quote marks may be used to delimit the prompt string, however, both the beginning and ending marks must be the same.

INPUT should be used to enter numeric and logical data only. The ACCEPT command is a more convenient way to enter character strings.

Examples:

```
. INPUT TO X
:3
  3

. INPUT TO Z
:23/17.000+X
 4.352

. INPUT 'PROMPT USER FOR INPUT' TO Q
PROMPT USER FOR INPUT:12345
  12345

. INPUT 'ENTER T IF EVERYTHING IS OKAY' TO LOG
ENTER T IF EVERYTHING IS OKAY:T
.T.

. INPUT "ENTER A CHAR STRING" TO CHAR
ENTER A CHAR STRING:'CHAR STRING MUST BE QUOTE DELIMITED'
CHAR STRING MUST BE QUOTE DELIMITED
```

```
. DISP MEMO
X           (N)   3
Z           (N)   4.352
Q           (N)   12345
LOG         (L)  .T.
CHAR        (C)  CHAR STRING MUST BE QUOTE DELIMITED
** TOTAL **       05 VARIABLES USED  00054 BYTES USED

. INPUT 'ENTER ANY LOGICAL ' TO LOG2
ENTER ANY LOGICAL :y
.T.
```

INSERT

```
INSERT [BEFORE] [BLANK]
```

This command allows records to be INSERTed into the middle of a database. Only one record at a time may be inserted into the database with the INSERT command.

The BEFORE phrase is used to cause insertion before the record currently pointed at, otherwise the new record will be placed just after the current record. Unless the BLANK phrase is used, the user will be prompted for input values as with the APPEND and CREATE commands. If the BLANK phrase is specified, then an empty record is inserted.

If the CARRY is SET ON then the information in the previous record is carried over to the new record.

INSERTs into a large non-indexed database take a long time to complete and should be avoided unless necessary. INSERTs into an indexed file, no matter what size, are identical to APPENDs.

Examples:

```
. USE SHOPLIST

. LIST
00001  BEANS #303 CAN          5        0.69
00002  BREAD LOAVES            2        0.89
00003  T-BONE STEAKS           4        3.59
00004  LETTUCE                 2        0.49
00005  MILK (1/2 GAL)          2        1.19
00006  CHARCOAL, 5# BAGS       2        0.69

. GOTO RECORD 4

. INSERT

RECORD 00005

ITEM:        BLEU CHEESE
NO:          1
COST:        1.79
```

```
. LIST
00001  BEANS #303 CAN          5        0.69
00002  BREAD LOAVES            2        0.89
00003  T-BONE STEAKS           4        3.59
00004  LETTUCE                 2        0.49
00005  BLEU CHEESE             1        1.79
00006  MILK (1/2 GAL)          2        1.19
00007  CHARCOAL, 5# BAGS       2        0.69

. GOTO RECORD 4

. INSERT BEFORE

RECORD 00004

ITEM:        PAPER PLATES
NO:          1
COST:        .79

. LIST
00001  BEANS #303 CAN          5        0.69
00002  BREAD LOAVES            2        0.89
00003  T-BONE STEAKS           4        3.59
00004  PAPER PLATES            1        0.79
00005  LETTUCE                 2        0.49
00006  BLEU CHEESE             1        1.79
00007  MILK (1/2 GAL)          2        1.19
00008  CHARCOAL, 5# BAGS       2        0.69

. 4

. DISPLAY
00004  PAPER PLATES            1        0.79

. INSERT BLANK

. LIST
00001  BEANS #303 CAN          5        0.69
00002  BREAD LOAVES            2        0.89
00003  T-BONE STEAKS           4        3.59
00004  PAPER PLATES            1        0.79
00005
00006  LETTUCE                 2        0.49
00007  BLEU CHEESE             1        1.79
00008  MILK (1/2 GAL)          2        1.19
00009  CHARCOAL, 5# BAGS       2        0.69
```

```
. 5

. REPLACE ITEM WITH 'PLASTIC FORKS' AND NO WITH 5 AND COST
WITH .39

00001 REPLACEMENT(S)

. LIST
00001  BEANS #303 CAN          5        0.69
00002  BREAD LOAVES            2        0.89
00003  T-BONE STEAKS           4        3.59
00004  PAPER PLATES            1        0.79
00005  PLASTIC FORKS           5        0.39
00006  LETTUCE                 2        0.49
00007  BLEU CHEESE             1        1.79
00008  MILK (1/2 GAL)          2        1.19
00009  CHARCOAL, 5# BAGS       2        0.69
```

JOIN

JOIN TO <file> FOR <expression> [FIELDS <field list>]

This is one of the most powerful commands in dBASE. It allows two databases to be JOINed together to form a third database whenever some criterion is met.

The two databases used are the primary and secondary USE files. First the SELECT PRIMARY command is issued. Then the JOIN command is issued. JOIN then positions dBASE to the first record of the primary USE file and evaluates the ON expression for each record in the secondary USE file. Each time that the expression yields a TRUE result, a record as added TO the new database. When the end of the secondary USE file is reached, the primary USE file is advanced one record, the secondary USE file is 'rewound' and the process continues until the primary USE file is exhausted.

If the FIELDS phrase is omitted then the output database will be comprised of all the fields in the primary USE file's structure and as many of the secondary USE file's fields as will fit before exceeding the 32 field limit of dBASE.

If the FIELDS phrase is supplied, then those fields, and only those fields, that are in the field list will be placed in the output database.

This command takes a lot of time to complete if the contributing databases are large. And if the joining criterion is too loose, causing many joinings per primary record, then there is the potential for causing a JOIN that dBASE cannot complete. For example, suppose that the primary and secondary USE files each contain a 1000 records, and that the expression is always true, a million records should be output by the JOIN into a database whose size would exceed the dBASE maximum of 65,535 records.

Example:

```
.USE INVENTRY

.DISPLAY STRUCTURE
STRUCTURE FOR FILE:    INVENTRY.DBF
NUMBER OF RECORDS:     00008
DATE OF LAST UPDATE:   00/00/00
PRIMARY USE DATABASE
FLD        NAME       TYPE WIDTH  DEC
001      ITEM          C    020
002      COST          N    010    002
003      PART:NO       C    005
004      ON:HAND       N    005
** TOTAL **               00041

. LIST
00001  TIME STITCH                 9.99 24776     1
00002  WIDGET                      1.67 31415    18
00003  GADGET, LARGE              16.33 92653     7
00004  TANK, SHERMAN          134999.00 89793     3
00005  SINK, KITCHEN              34.72 21828    77
00006  THROMBONES                198.37 76767    76
00007  RINGS, GOLDEN             200.00 70296     5
00008  #9 COAL                    22.00 11528    16

. SELECT SECONDARY

. USE ORDERS

. DISPLAY STRUCTURE
STRUCTURE FOR FILE:    ORDERS.DBF
NUMBER OF RECORDS:     00008
DATE OF LAST UPDATE:   00/00/00
PRIMARY USE DATABASE
FLD       NAME       TYPE WIDTH  DEC
001     CUSTOMER      C    020
002     PART:NO       C    005
003     AMOUNT        N    005
** TOTAL **              00031

. LIST
00001  SWARTZ, JOE             31415     13
00002  SWARTZ, JOE             76767     13
00003  HARRIS, ARNOLD          11528     44
00004  ADAMS, JEAN             89793     12
00005  MACK, JAY               31415      3
00006  TERRY, HANS             76767      5
00007  JUAN, DON               21828      5
00008  SALT, CLARA             70296      9
```

```
. SELECT PRIMARY

. JOIN TO ANNOTATE FOR PART:NO=S.PART:NO;
FIELD CUSTOMER,ITEM,AMOUNT,COST
```

use the inventory file to add names to the orders

```
. USE ANNOTATE

. DISPLAY STRUCTURE
STRUCTURE FOR FILE:    ANNOTATE.DBF
NUMBER OF RECORDS:     00008
DATE OF LAST UPDATE:   00/00/00
PRIMARY USE DATABASE
FLD        NAME       TYPE WIDTH  DEC
001      CUSTOMER      C    020
002      ITEM          C    020
003      AMOUNT        N    005
004      COST          N    010     002
** TOTAL **               00056

. LIST
00001  SWARTZ, JOE          WIDGET                 13       1.67
00002  MACK, JAY            WIDGET                  3       1.67
00003  ADAMS, JEAN          TANK, SHERMAN          12  134999.00
00004  JUAN, DON            SINK, KITCHEN           5      34.72
00005  SWARTZ, JOE          TROMBONES              13     198.37
00006  TERRY, HANS          TROMBONES               5     198.37
00007  SALT, CLARA          RINGS, GOLDEN           9     200.00
00008  HARRIS, ARNOLD       #9 COAL                44      22.00

. USE INVENTRY
```

(join customer names with part numbers with insuffcent inventory to satisfy orders so that the customers can be notified, for instance)

```
. JOIN TO BACKORDR FOR PART:NO=S.PART:NO.AND.ON:HAND<AMOUNT;
FIELD CUSTOMER,ITEM

. USE BACKORDR

. LIST
00001  ADAMS, JEAN          TANK, SHERMAN
00002  SALT, CLARA          RINGS, GOLDEN
00003  HARRIS, ARNOLD       #9 COAL
```

LIST

LIST is the same as DISPLAY, except the scope defaults to ALL records and WAIT does not wait for a go-ahead after 15 record groups. Notice however that LIST STRUCTURE, LIST FILES and LIST MEMORY commands work exactly as the DISPLAY command.

LOCATE

```
LOCATE [<scope>] [FOR <exp>]
[CONTINUE]
```

This command causes a search of database records in the USE file for the first record whose data fields allow the expression <exp> to be TRUE. When the expression is satisfied, the following message is displayed:

```
RECORD n
```

The CONTINUE command may be used to continue the search. Other dBASE commands may be issued between the LOCATE and the CONTINUE. This does, however, limit the number of the characters in the FOR <exp> to 128 instead of 254. See CONTINUE.

If the expression cannot be found, the message END OF FILE is displayed, and the database is left positioned at the last record in the file. If the NEXT clause (see scope, section 9.1) is used in this command and the expression cannot be found within the scope of the NEXT, the message END OF LOCATE is displayed, and the database is left positioned at the last record scanned.

Note: a LOCATE will work faster on a file that is USEd without an INDEX file.

Examples:

```
 . USE SHOPLIST

 . LIST
 00001  BEANS #303 CAN          5       0.69
 00002  BREAD LOAVES            2       0.89
 00003  T-BONE STEAKS           4       3.59
 00004  PAPER PLATES            1       0.79
 00005  PLASTIC FORKS           5       0.39
 00006  LETTUCE                 2       0.49
 00007  BLEU CHEESE             1       1.79
 00008  MILK (1/2 GAL)          2       1.19
 00009  CHARCOAL, 5# BAGS       2       0.69

 . LOCATE FOR COST>.70
 RECORD: 00002

 . CONTINUE
 RECORD: 00003

 . DISP ITEM
 T-BONE STEAKS
 . CONTINUE
 RECORD: 00004
```

```
. CONTINUE
RECORD: 00007

. CONTINUE
RECORD: 00008

. CONTINUE
END OF FILE
```

LOOP

LOOP

This command is used within the body of a DO WHILE to skip the commands following the LOOP, and still allow the reappraisal and possible reexecution of the body of the DO WHILE. LOOP is used to shorten DO WHILE loops which, if large, can be time consuming or may contain commands which are to be skipped at times. LOOP acts much as an ENDDO command, it will backup to the DO WHILE that matches it in nesting depth.

Use of loops in a DO WHILE is not a good programming practice and should be avoided. The following example was done a second time, the second follows the first, without use of the LOOP capability.

Example:

```
STORE 1 TO INDEX
DO WHILE INDEX<10
  STORE INDEX+1 TO INDEX
  IF ITEM='   '                 Anytime that ITEM is equal to blanks
    SKIP                        then skip to the next record
    LOOP                        and go back to the DO WHILE
  ENDIF
  DO PROCESS
ENDDO
```

Example 2:

```
STORE 1 TO INDEX
DO WHILE INDEX < 10
  STORE INDEX + 1 TO INDEX
  IF ITEM = '   '
    SKIP
  ELSE
    DO PROCESS
  ENDIF
ENDDO
```

MODIFY

a. MODIFY STRUCTURE
b. MODIFY COMMAND [<command file>]

Form a. of this command allows the user to modify the structure of a dBASE file. Any changes are permitted. Fields can be added, deleted, or have their parameters (e.g. name, type, length, number of decimals) changed.

MODIFY acts upon the database currently in USE. The existing structure is displayed on the screen, changes are made directly on the screen in the same way as full-screen editing is done with two exceptions: CTL-N inserts a blank line wherever the cursor is, CTL-T deletes the line that the cursor is on. The other control keys behave as described in section 9.

NOTE: the MODIFY STRUCTURE command deletes ALL data records that were in the USE file prior to the MODIFY. In order to modify a structure and keep its data, first COPY the structure to a work file, USE the work file, make the modifications, and finally APPEND the old data to the work file. The original database and the work file may be RENAME'd if it is necessary to restore their original names. See the example below.

Form b. of this command allows minor full-screen editing of command files (or anything else). If the <command file> is omitted then the user is prompted for it. If the file doesn't exist, it is created. After a command file has been edited, MODIFY COMMAND will rename type of the old copy to .BAK and save the new copy with the type .CMD.

When in MODIFY COMMAND, the CTL-N and CTL-T editing functions work as described in a previous paragraph. CTL-Q will abort all changes to the command file, CTL-W will write the changes back to the disk and to the rename that was described above.

There are some significant restrictions to this form of the command: 1) lines can only be 77 or fewer characters long (including the carriage return/line feed pair); 2) TAB characters are converted to single spaces; 3) the cursor can only be backed up in a file about 4000 bytes; 4) there is no search or block move capability as are in some text editors.

Full-screen cursor controls are the same for MODIFY COMMAND **EXCEPT** for the following commands:

ctl-N	-	inserts a blank line wherever the cursor is;
ctl-T	-	deletes the line the cursor is on and moves up the lower lines;
ctl-W	-	writes the changes made to the file back on the disk and exits MODIFY COMMAND (ctl-o for SuperBrain);
ctl-Q	-	aborts any changes made to the command file;
ctl-R	-	scrolls one line down; and
ctl-C	-	scrolls one page up.

Example:

```
. NOTE -- AN EXAMPLE OF HOW TO MODIFY A STRUCTURE WITHOUT

. NOTE    LOSING THE INFORMATION IN THE FILE

. USE INVNTRY

. COPY TO WORK

. USE WORK

. MODIFY STRUCTURE

. APPEND FROM INVNTRY

. DELETE FILE INVNTRY

. USE

. RENAME WORK TO INVNTRY
```

NOTE
----

a. NOTE any characters
b. * any characters

This command allows comments to be placed into a command file. Unlike the REMARK command, the content of this command is not echoed onto the output device.

Example:

```
NOTE - last modification : 4 july 1976

* -- last modification spelled doom's day
```

PACK

PACK

This command purges all records marked for deletion by the DELETE command. Once the PACK command has been issued, nothing can bring back deleted records.

If the file being PACKed is indexed, and the indexed file is in use, then the PACK will adjust the index file at the same time it adjusts the USE file. For large indexed files, doing a PACK on the file without the index and then reindexing is faster.

If the database is indexed by more that one index file, then the other index files must be reINDEXed on those keys since the PACK will (in all probability) have moved records around.

An alternate method to the PACK is to COPY the old file to a new file. DELETEd records will not be copied. Then the old file may be deleted (or saved as a back-up) and the new file renamed.

Examples:

```
. USE B:SHOPSAVE

. LIST
00001  BEANS              5      0.75
00002  BREAD LOAVES       2      0.97
00003  T-BONE             4      3.94
00004  PAPER PLATES       1      0.86
00005  PLASTIC FORKS      5      0.42
00006  LETTUCE            2      0.53
00007  BLEU CHEESE        1      1.96
00008  MILK               2      1.30
00009  CHARCOAL           2      0.75

. DELETE RECORD 8
00001 DELETION(S)

. LIST
00001  BEANS              5      0.75
00002  BREAD LOAVES       2      0.97
00003  T-BONE             4      3.94
00004  PAPER PLATES       1      0.86
00005  PLASTIC FORKS      5      0.42
00006  LETTUCE            2      0.53
00007  BLEU CHEESE        1      1.96
00008 *MILK               2      1.30
00009  CHARCOAL           2      0.75

. PACK
PACK COMPLETE, 00008 RECORDS COPIED
```

```
. LIST
00001  BEANS                5      0.75
00002  BREAD LOAVES         2      0.97
00003  T-BONE               4      3.94
00004  PAPER PLATES         1      0.86
00005  PLASTIC FORKS        5      0.42
00006  LETTUCE              2      0.53
00007  BLEU CHEESE          1      1.96
00008  CHARCOAL             2      0.75
```

A PACK need not always be done, for example, suppose some records must be deleted but it is necessary for them to remain in the database. These records will not be COPY'd, APPENDed, or SORTed; they will however be COUNTed. It becomes important to know wether or not the record being processed is deleted or not. The following example is a partial command file that would skip over a record that has been deleted and continue processing with the next record.

```
DO WHILE .NOT. EOF
  LOCATE FOR NATURE = "TLM"
  IF .NOT. *
    .
    commands
    .
  ENDIF
  CONTINUE
ENDDO
```

QUIT

QUIT [TO <com file list>]

This command closes all database files, command files, and alternate files and returns control to the operating system. The message *** END RUN dBASE *** is displayed.

If the TO phrase is included, then all the programs in the <com file list> will be executed in sequence by CP/M. This feature lets you to go out of dBASE and chain to other pieces of software.

There is no limit to the number of programs or CP/M commands which can be executed as long as the 254 character limit for any command is not broken. dBASE be reentered an the end of the string of commands. However, it is not required; CP/M will be given control when the string of commands are all finished executing.

Example:

```
   QUIT TO 'DIR B:','PIP PRN:=ALTERNAT.TXT','DBASE CMDFILE'
```

In this example, dBASE is exited, a directory of the B-drive is done, PIP is then called to copy a file to the print device, and dBASE is reentered with a command file (CMDFILE.CMD) taking control immediately.

READ

READ

This command enters the full-screen mode for editing and/or data entry of variables identified for and displayed by an "@" command with a GET phrase. The cursor can be moved to any of the GET variables. Changes made to those variables on the screen are entered into the appropiate database fields or memory variables.

If the SET FORMAT TO <format file> command has been issued, then READ will cause all of the "@" commands in the format file to be executed, thus formatting the screen, allowing editing of all GET variables. Notice that this technique is a tailorable substitute for the EDIT command when in the interactive mode.

When in the SET FORMAT TO SCREEN mode, an ERASE command is used to clear the screen. A series of "@" commands may then be issued to format the screen. Then a READ command would be given which would allow editing.

If a second or later series of "@" commands is issued after a READ command, then READ will place the cursor on the first GET variable following the last READ. In this way, the screen format and the specific variables edited can be based on decisions made by the user in response to prior READ commands.

Variables to be used with the "@" commands and edited using the READ command must be either in the USE file as field names or must be character string memory variables. Memory variables must be predefined before the "@" command is issued. If necessary, store as many blanks as you want the maximum length of the memory variable to be in order to initialize the memory variable (e.g. STORE '   ' to MEMVAR).

See section 8 for cursor control and data entry instructions.

The SET SCREEN ON command must be in effect (this is the default condition if full-screen operations were enabled when dBASE II was installed).

Example:

```
.
.
.
STORE ' ' TO PTYPE
STORE '           ' TO ACCT
ERASE
@ 5,0 SAY 'Enter a C for cash payment'
@ 6,0 SAY '   or a D for deferred payment'
@ 8,10 GET PTYPE
READ
IF PTYPE='D'
   @ 10,10 SAY 'Enter acct no.' GET ACCT PICTURE '999-99-9999'
   READ
ENDIF
.
.
.
```

In this command file fragment, the screen is cleared and the first two "@" commands are put up. The cursor will be between two colons that mark the screen location of the variable PTYPE. Since the first STORE set the size of PTYPE at 1 character, any entry by the user will fill PTYPE and exit the first READ command.

If a "D" was entered by the dBASE operator, then the "@" command that asks for an account number will be done. Notice that ACCT was defined long enough in the STORE to include the two dashes that the PICTURE phrase in the "@" will enter

```
USE CHECKS
SET FORMAT TO SCREEN
ACCEPT "Option" TO CHOICE
IF CHOICE$'Aa'
  ERASE
  DO WHILE NUMBER # 0
    APPEND BLANK
    @ 5,0 SAY "Enter next Number" ;
          GET NUMBER PICTURE '99999'
    @ 6,0 SAY "Enter Recipient";
          GET RECIPIENT PICTURE 'XXXXXXXXXXXXXXXXXXXXXXXX'
    @ 7,0 SAY "Enter Amount";
          GET AMOUNT PICTURE '9999999999'
    @ 8,5 SAY "Is it back yet?" ;
          GET HOME
    @ 8,30 SAY "Are you paying out?";
          GET OUTGOING
    READ
  ENDDO
ENDIF
```

In the last example, a file was used and altered directly, the choice being left up to the operator on whether or not to add new records to the database in question.

Refer to the "@" command for more details.

RECALL

RECALL [<scope>] [FOR <exp>]

This command removes the mark-for-deletion from the records that were marked by the DELETE command.

Examples:

```
. USE DUPE3

. LIST
00001  NEUMAN, ALFRED E.    1357
00002  RODGERS, ROY         2468
00003  CASSIDY, BUTCH       3344
00004  CHANG, LEE           6743
00005  POST, WILEY          1011
00006  LANCASTER, WILLIAM J 6623

. 3

. DELETE NEXT 3
00003 DELETION(S)

. LIST
00001  NEUMAN, ALFRED E.    1357
00002  RODGERS, ROY         2468
00003 *CASSIDY, BUTCH       3344
00004 *CHANG, LEE           6743
00005 *POST, WILEY          1011
00006  LANCASTER, WILLIAM J 6623

. RECALL RECORD 4
00001 RECALL(S)

. LIST
00001  NEUMAN, ALFRED E.    1357
00002  RODGERS, ROY         2468
00003 *CASSIDY, BUTCH       3344
00004  CHANG, LEE           6743
00005 *POST, WILEY          1011
00006  LANCASTER, WILLIAM J 6623

. RECALL ALL
00002 RECALL(S)
```

```
. LIST
00001  NEUMAN, ALFRED E.    1357
00002  RODGERS, ROY         2468
00003  CASSIDY, BUTCH       3344
00004  CHANG, LEE           6743
00005  POST, WILEY          1011
00006  LANCASTER, WILLIAM J 6623
```

(This page is left intentionally blank)

(This page is left intentionally blank)

RELEASE

```
RELEASE [<memvar list>]
        [ALL]
```

This command releases all or selected memory variables and makes the space that they consumed available for new memory variables. If ALL is specified, then all memory variables will be deleted.

REMARK

REMARK any characters.

This command allows the display of any characters. The contents of this command are displayed on the output device when this command is encountered.

Examples:

```
. REMARK ****** REMARK TEST ********
 ****** REMARK TEST ********
```

RENAME

RENAME <original file name> TO <new file name>

This command allows the changeing of the name of a file in the CP/M directory. If no file type (the up to 3 characters following a file name) is given then dBASE assumes that a database's name is being used and assigns the type .DBF to the named files. See section 4 for more detail concerning dBASE use of file types.

Example:

```
. RENAME INVENMAC TO INVENOLD

. RENAME D:REPORT.FRM TO REPORT.BAK

. RENAME TYPELESS. TO TYPED.TYP
```

REPLACE

```
REPLACE [<scope>] <field> WITH <exp> [,<field2> WITH <exp2>] ,etc
            [FOR <exp>]
```

This command is used to replace the contents of specified data fields of the file in USE with some new data. This command is contrasted with the STORE command in that REPLACE changes only field variables, while the STORE command changes only memory variables.

If <scope> is not supplied in the command then REPLACE acts only on the current record.

If a REPLACE is done on an index key and the index is in USE, then the index file will be adjusted by deleting the old index entry and re-entering the new entry in its proper place. Un-USEd index files will not be affected. When a REPLACE is done on an index key, the altered record will "shift places" in the file, the new "next record" will not be the same as the old "next record". The key should not be REPLACEd with a NEXT n as the <scope>.

Examples:

```
. USE SHOPLIST

. NOTE INFLATION CAUSES 10% PRICE INCREASE

. LIST
00001  BEANS #303 CAN          5        0.69
00002  BREAD LOAVES            2        0.89
00003  T-BONE STEAKS           4        3.59
00004  PAPER PLATES            1        0.79
00005  PLASTIC FORKS           5        0.39
00006  LETTUCE                 2        0.49
00007  BLEU CHEESE             1        1.79
00008  MILK (1/2 GAL)          2        1.19
00009  CHARCOAL, 5# BAG.       2        0.69

. REPLACE ALL COST WITH COST*1.1
00009 REPLACEMENT(S)
```

```
. LIST
00001  BEANS #303 CAN          5        0.75
00002  BREAD LOAVES            2        0.97
00003  T-BONE STEAKS           4        3.94
00004  PAPER PLATES            1        0.86
00005  PLASTIC FORKS           5        0.42
00006  LETTUCE                 2        0.53
00007  BLEU CHEESE             1        1.96
00008  MILK (1/2 GAL)          2        1.30
00009  CHARCOAL, 5# BAGS       2        0.75

. USE B:SHOPLIST

. COPY TO B:SHOPWORK
00009 RECORDS COPIED

. LIST
00001  BEANS                   5        0.75
00002  BREAD LOAVES            2        0.97
00003  T-BONE                  4        3.94
00004  PAPER PLATES            1        0.86
00005  PLASTIC FORKS           5        0.42
00006  LETTUCE                 2        0.53
00007  BLEU CHEESE             1        1.96
00008  MILK                    2        1.30
00009  CHARCOAL                2        0.75

. GOTO TOP

. REPLACE NEXT 5 COST WITH COST*1.1 FOR COST>.75
00003 REPLACEMENT(S)

. LIST
00001  BEANS                   5        0.75
00002  BREAD LOAVES            2        1.06
00003  T-BONE                  4        4.33
00004  PAPER PLATES            1        0.94
00005  PLASTIC FORKS           5        0.42
00006  LETTUCE                 2        0.53
00007  BLEU CHEESE             1        1.96
00008  MILK                    2        1.30
00009  CHARCOAL                2        0.75

. USE CHECKS

. DISP STRU
```

```
STRUCTURE FOR FILE:   CHECKS.DBF
NUMBER OF RECORDS:    00016
DATE OF LAST UPDATE: 10/18/81
PRIMARY USE DATABASE
FLD        NAME      TYPE WIDTH    DEC
001       NUMBER      N    005
002       RECIPIENT   C    020
003       AMOUNT      N    010     002
004       HOME        L    001
005       OUTGOING    L    001
** TOTAL **              00038

. LIST

00001        1 Phone Company              104.89 .F. .T.
00002        2 Gas Company                  4.15 .F. .T.
00003        3 Electricity                250.30 .F. .T.
00004        4 Grocery Store             1034.45 .F. .T.
00005      134 Me, salary                 561.77 .T. .F.
00006        6 Bank (sc)                    4.00 .T. .T.
00007        7 Doctor Doolittle           100.00 .T. .T.
00008        8 Pirates                    100.00 .F. .T.
00009        9 Car Repair Man             500.01 .F. .T.
00010       10 Me                         561.77 .T. .F.
00011       11 Tuperware                   50.02 .F. .T.
00012       12 Me                         561.77 .T. .F.
00013       13 Me                         750.03 .T. .F.
00014      234 Peter Rabbit                14.00 .F. .T.
00015      237 Golden Goose               650.00 .F. .T.
00016       30 Me                         561.77 .T. .F.

. 11

. REPLACE HOME WITH F
00001 REPLACEMENT(S)

. DISPLAY
00011       11 Tuperware                   50.02 .F. .T.
```

REPORT

REPORT [FORM <form file>] [<scope>] [TO PRINT] [PLAIN]

REPORT is used to prepare reports (either on the screen or on paper) by displaying data from the file in USE in a defined manner. Reports may have titled columns, totaled numeric fields, and displayed expressions involving data fields, memory variables, and constants.

The FOR phrase allows only that information which meets the conditions of the <exp> to be reported; the TO PRINT phrase sends the report to the printer as well as the screen; and the <scope> of the report defaults to ALL unless otherwise specified.

The first time the REPORT command is used (for a new report) a FORM file is built. dBASE prompts the user for specifications of the report format and automatically generates the FORM file. Subsequent reports can use the FORM file to avoid respecification of the report format. If the FORM phrase of the command is omitted the user will be prompted for the name of the form file.

The following example of a form file has almost all the options specified. The user may control the number of spaces to indent the lines in the body of the report with the 'M' option (default is 8 spaces); the number of lines per page is changed with the 'L' option (default is 57 lines); and the location of the page heading is controlled with the 'W' option (the page width, default is 80 characters) since it is only used for centering the page heading.

```
. REPORT FORM SHOPFORM
ENTER OPTIONS, M=LEFT MARGIN, L=LINES/PAGE, W=PAGE WIDTH M=5,W=65
PAGE HEADING? (Y/N) Y
ENTER PAGE HEADING: Shopping List for Picnic
DOUBLE SPACE REPORT? (Y/N) N
ARE TOTALS REQUIRED? (Y/N) Y
SUBTOTALS IN REPORT? (Y/N) N
COL      WIDTH,CONTENTS
001       23,ITEM+'...'
ENTER HEADING: Item;====
002       10,NO
ENTER HEADING: >Number;======
ARE TOTALS REQUIRED? (Y/N) Y
003       10,COST
ENTER HEADING: >Cost/Item;=========
ARE TOTALS REQUIRED? (Y/N) N
004       10,NO*COST
ENTER HEADING: >COST;====
ARE TOTALS REQUIRED? (Y/N) Y
005       (cr)
```

REPORT asks for the width of the field to be printed and the contents of the field. The width asked for here has no relationship to the actual width of the field to be printed out, for instance, in the first column above, ITEM is in a column that is 23 characters wide, in the data base ITEM is actually only 20 characters wide. One should also note that the string '...' is being concatenated to the contents of the field ITEM. This accounts for the extra 3 characters in the report. This also means that if the report column is less in length than the field that should go into it, dBASE will wrap the field to fit. An 80 character field would generate 2 lines if it were put into a 50 character column.

The contents of the columns may be fields from a database, a memory variable, literals, or expressions. Note that in column 1 in the form on the previous page, there is a concatenated string. Each record in the database in use will have <u>only as far as the report is concerned (the database will remain unchanged)</u> three periods concatenated to the end of the string. Column 4 contains the product of NO and COST. Column 4 has no field equivalent to it in the database. (The fields are, left to right, named ITEM, NO, and COST)

```
. LIST
00001  BEANS                    5      0.75
00002  BREAD LOAVES             2      1.06
00003  T-BONE                   4      4.33
00004  PAPER PLATES             1      0.94
00005  PLASTIC FORKS            5      0.42
00006  LETTUCE                  2      0.53
00007  BLEU CHEESE              1      1.96
00008  MILK                     2      1.30
00009  CHARCOAL                 2      0.75
```

Returning to the FORM file (the questions on what should go into the report), note that there are some special characters used in the headings. For page headings, column headings, and character strings, a semicolon (;) will break the heading or string at the semicolon and resume the display on the next line. If a heading or string is too long to fit within the number of spaces allowed for it, it will be broken at the last blank (if possible) and resumed on the next line. The other signifigant characters are "<", and ">". In column headings, if the title is preceeded with a "<" then the title will be left-justified in the column. Likewise a ">" will right-justify the title.

Other options in REPORT include totalling, subtotalling, and summary reports. In summary reports, detail records are not displayed, just totals and subtotals. Totalling and subtotalling is done only on fields that are numeric in nature. See the report examples.

Finally a carriage return will end the report form and begin displaying the report. A copy will be printed on the printer if the TO PRINT phrase was included in the initial command.

Other dBASE commands that effect the operation of report are the "SET EJECT OFF", "SET HEADING TO" and "SET DATE TO" commands. Before REPORT prints out its information, it does a page eject. This capability may be suppressed with the SET EJECT OFF command. The SET HEADING TO command allows an additional heading to be added to the report at run time. This command has an effect for the duration of one session. (The heading must be set each time a new dBASE run is initiated.) The same is for the SET DATE TO command. The date of the report may be changed or omitted by use of this command. See the SET command for more information.

There comes a time, when this capability is no longer adequate, special forms must be used, more flexability is desired with the report format, retrieving the data from the database requires more complex methods than REPORT will handle, etc. The "@" and the SET FORMAT TO PRINT commands will give the user more power over the form of the report. See the "@" command for more information and examples.

Examples:

```
. USE SHOPLIST

. REPORT FORM SHOPFORM


   PAGE NO. 00001

                        Shopping List for Picnic

              Item                     Number  Cost/Item       COST
              ====                     ======  =========       ====

   BEANS                 ...              5      0.75          3.75
   BREAD LOAVES          ...              2      1.06          2.12
   T-BONE                ...              4      4.33         17.32
   PAPER PLATES          ...              1      0.94          0.94
   PLASTIC FORKS         ...              5      0.42          2.10
   LETTUCE               ...              2      0.53          1.06
   BLEU CHEESE           ...              1      1.96          1.96
   MILK                  ...              2      1.30          2.60
   CHARCOAL              ...              2      0.75          1.50
   ** TOTAL **
                                         24                   33.35
```

```
. SET HEADING TO 4 July 1976

. REPORT FORM SHOPFORM

     PAGE NO. 00001         4 July 1976

                         Shopping List for Picnic

                    Item             Number  Cost/Item        COST
                    ====             ======  =========        ====

     BEANS                 ...            5       0.75        3.75
     BREAD LOAVES          ...            2       1.06        2.12
     T-BONE                ...            4       4.33       17.32
     PAPER PLATES          ...            1       0.94        0.94
     PLASTIC FORKS         ...            5       0.42        2.10
     LETTUCE               ...            2       0.53        1.06
     BLEU CHEESE           ...            1       1.96        1.96
     MILK                  ...            2       1.30        2.60
     CHARCOAL              ...            2       0.75        1.50
     ** TOTAL **
                                         24                  33.35
```

Example 2:

This example shows use of the subtotalling capabilities of dBASE. When the report form is created the subtotalling is done on the field PART:NO. This could be done if it was necessary to know not only who the part was ordered by but also how many of each part must be made (or bought).

```
 . USE ORDERS INDEX ORDERS

 . LIST
00003  HARRIS, ARNOLD          11528    44
00013  ANDERSON, JAMES REGI    11528    16
00007  JUAN, DON               21828     5
00001  SWARTZ, JOE             31415    13
00005  MACK, JAY               31415     3
00009  BARNETT, WALT           31415     6
00008  SALT, CLARA             70296     9
00002  SWARTZ, JOE             76767    13
00006  TERRY, HANS             76767     5
00010  NICHOLS, BILL           76767    17
00004  ADAMS, JEAN             89793    12
00011  MURRAY, CAROL           89793     4
00012  WARD, CHARLES A.        92653    15
```

```
. REPORT
ENTER REPORT FORM NAME: ORDERS
ENTER OPTIONS, M=LEFT MARGIN, L=LINES/PAGE, W=PAGE WIDTH W=65
PAGE HEADING? (Y/N) Y
ENTER PAGE HEADING: ORDERS LISTED BY PART NUMBER
DOUBLE SPACE REPORT? (Y/N) N
ARE TOTALS REQUIRED? (Y/N) Y
SUBTOTALS IN REPORT? (Y/N) Y
ENTER SUBTOTALS FIELD: PART:NO
SUMMARY REPORT ONLY? (Y/N) N
EJECT PAGE AFTER SUBTOTALS? (Y/N) N
ENTER SUBTOTAL HEADING: Orders for part number
COL     WIDTH,CONTENTS
001     20,CUSTOMER
ENTER HEADING: <CUSTOMER NAME
002     10,AMOUNT
ENTER HEADING: >QUANTITY ORDERED
ARE TOTALS REQUIRED? (Y/N) Y
003


         PAGE NO. 00001

                         ORDERS LISTED BY PART NUMBER

         CUSTOMER NAME           QUANTITY
                                  ORDERED

         * Orders for part number 11528
         HARRIS, ARNOLD                44
         ANDERSON, JAMES REGI          16
         ** SUBTOTAL **
                                       60


         * Orders for part number 21828
         JUAN, DON                      5
         ** SUBTOTAL **
                                        5


         * Orders for part number 31415
         SWARTZ, JOE                   13
         MACK, JAY                      3
         BARNETT, WALT                  6
         ** SUBTOTAL **
                                       22
```

```
* Orders for part number 70296
SALT, CLARA                     9
** SUBTOTAL **

                                9

* Orders for part number 76767
SWARTZ, JOE                    13
TERRY, HANS                     5
NICHOLS, BILL                  17
** SUBTOTAL **

                               35

* Orders for part number 89793
ADAMS, JEAN                    12
MURRAY, CAROL                   4
** SUBTOTAL **

                               16

* Orders for part number 92653
WARD, CHARLES A.               15
** SUBTOTAL **

                               15

** TOTAL **

                              162
```

Example 3:

Suppose some of your collegues and yourself started playing cards for points to see who would buy lunch for everyone on the next holiday. In the interest of Fair Play, you decide to keep a running total on the score. All sorts of information could be dug out of the database (like who could loqse his shirt if he didn't be careful). The following database could be an example of such a game.

```
. DISP STRU
STRUCTURE FOR FILE:   CARDS.DBF
NUMBER OF RECORDS:    00016
DATE OF LAST UPDATE:  09/17/81
PRIMARY USE DATABASE
FLD       NAME       TYPE WIDTH    DEC
001      DATE         C    008
002      LISA         N    003
003      ANNA         N    003
004      WAYNE        N    003
** TOTAL **              00018
```

```
. REPORT
ENTER REPORT FORM NAME: CARDS
ENTER OPTIONS, M=LEFT MARGIN, L=LINES/PAGE, W=PAGE WIDTH W=40
PAGE HEADING? (Y/N) Y
ENTER PAGE HEADING: Hearts Scores
DOUBLE SPACE REPORT? (Y/N) N
ARE TOTALS REQUIRED? (Y/N) Y
SUBTOTALS IN REPORT? (Y/N) N
COL     WIDTH,CONTENTS
001      10,DATE
ENTER HEADING: Date of;Game
002      6,LISA
ENTER HEADING: Score;Lisa
ARE TOTALS REQUIRED? (Y/N) Y
003      6,ANNA
ENTER HEADING: Score;Anna
ARE TOTALS REQUIRED? (Y/N) Y
004      6,WAYNE
ENTER HEADING: Score;Wayne
ARE TOTALS REQUIRED? (Y/N) Y
005      5,LISA+ANNA+WAYNE
ENTER HEADING: Game;Total
ARE TOTALS REQUIRED? (Y/N) Y
006      (cr)
```

(Note--the last column in the report form is a totalling of the scores in each of the records, that is, the sum of Lisa's, Wayne's and Anna's scores. It is not necessary for the column in the report to exist in the database before it may be used, the field "LISA+ANNA+WAYNE" does not exist in the database "CARDS". This would be an example of how an expression may be placed in a report.)

```
PAGE NO. 00001

              Hearts Scores

  Date of   Score   Score   Score   Game
   Game     Lisa    Anna    Wayne   Total

05/26/81     29      75      53      157
05/27/81     45      48      63      156
05/28/81     50      56      74      180
05/29/81     86      24      72      182
06/05/81     43      12      75      130
06/12/81     42       9      27       78
06/26/81     84      35      63      182
07/06/81     33      71      26      130
08/19/81     37      55      38      130
09/15/81     19      57      54      130
09/16/81     15       7     108      130
09/17/81     59      13      58      130
** TOTAL **
            715     698     875     2288
```

A report may also cover just a few of the records in a file. Like:

- **GOTO RECORD 7**
- **REPORT NEXT 4 FORM CARDS**

```
PAGE NO. 00001

              Hearts Scores

  Date of   Score   Score   Score   Game
   Game     Lisa    Anna    Wayne   Total

07/07/81     40      63      27      130
07/09/81     55      41      60      156
07/13/81     40      63      54      157
07/23/81     38      69      23      130
** TOTAL **
            173     236     164      573
```

A report may also ask for information which would meet certain criteria. Like:

REPORT FORM CARDS FOR WAYNE < 50

```
PAGE NO. 00001

              Hearts Scores

  Date of   Score   Score   Score   Game
   Game     Lisa    Anna    Wayne   Total

06/12/81     42       9      27       78
07/06/81     33      71      26      130
07/07/81     40      63      27      130
07/23/81     38      69      23      130
08/19/81     37      55      38      130
** TOTAL **
            190     267     141      598
```

REPORT FORM NEXT WHILE CUSTOMER >="M"

```
PAGE NO. 00001
12/13/81


        CUSTOMER            PART  AMOUNT

MACK, JAY                   31415      3
MURRAY, CAROL               89793      4
NICHOLS, BILL               76767     17
SALT, CLARA                 70296      9
SWARTZ, JOE                 31415     13
SWARTZ, JOE                 76767     13
TERRY, HANS                 76767      5
WARD, CHARLES A.            92653     15
```

PLAIN is an extension of the command REPORT. This allows for a dBASE report to be created in such a manner that it may be inserted into a report generated by a wordprocessor.

The clause PLAIN causes page numbers and the date at the top of each page in the report to be suppressed. Page headings are inserted into the dBASE report only at the beginning of the report. If it is desired to suppress the page ejects between reports then the SET EJECT OFF must still be used.

Examples:

```
. USE TRACE INDEX DOC

. NOTE POSITION THE DATABASE AT THE FIRST RECORD FOR THE REPORT
. 304

. REPORT FORM TABLES PLAIN WHILE DOC = "3-280-T"
ENTER OPTIONS, M=LEFT MARGIN, L=LINES/PAGE, W=PAGE WIDTH
PAGE HEADING? (Y/N) Y
ENTER PAGE HEADING: TABLES
DOUBLE SPACE REPORT? (Y/N) N
ARE TOTALS REQUIRED? (Y/N) N
COL      WIDTH,CONTENTS
001      20,$(DOC,7,17)
ENTER HEADING: TABLE
002      40,DESCR
ENTER HEADING: REQUIREMENT
003      (cr)
```

```
                              TABLES

          TABLE                        REQUIREMENT

    Table 1              GLL Telemetry Modes
    Table 2              Allowable combinations of R/T and Record
                         Formats
    Table 2.3.2          Bus User Codes
    Table 3              GLL Bit rate allocation
    Table 4              Header Format
    Table 5              Format Identification
    Table 6              Commutation Map Identifier Assignment
    Table 7              S/C Clock Progression
    Table A2.2.1         Eng data layout
    Table A2.2.2         Fixed-Area Structure/Position
                         Identifiers
    Table A2.2.3         Variable Area Pocket Structure/Position
                         Identifier
    Table A2.2.4         CDS Fixed area Measurement Sampling Time
    Table A2.2.8         Engr Measurements
```

RESET

```
RESET
```

The RESET command is used to reset the CP/M bit map after a diskette has been swapped. Normally, if a diskette is swapped, CP/M will not allow writes to take place until after a warm or soft boot has taken place. RESET attempts to re-open all files which were open prior to the swap. If a file that was open is no longer mounted on an active disk drive, RESET closes the file internally.

WARNING: If a disk is swapped that contains a file with the same name as a file that was previously open, the RESET operation will erroneously not close that file. This condition can be avoided by closing all non-essential files prior to the swap and subsequent RESET command. A USE command with no filename will close the file in USE, a CANCEL command will close any command files that may be open.

Issuing a RESET command when no disk swap has taken place has no effect.

RESTORE

```
RESTORE FROM <file>
```

This command reads a file of memory variables. The file must be built using the SAVE MEMORY TO <file> command. All memory variables which were defined previous to the RESTORE command are deleted by this command.

Examples:

```
. DISPLAY MEMORY
ONE         (N)   1.0000
ALFABET     (C)  ABCDEFGHIJKL
CHARS       (C)  ABCDEFGHIJKL NEW STUFF
** TOTAL **       03 VARIABLES USED  00042 BYTES USED

. SAVE TO MEMFILE

. RELEASE ALL

. DISPLAY MEMORY
** TOTAL **       00 VARIABLES USED  00000 BYTES USED

. RESTORE FROM MEMFILE

. DISPLAY MEMORY
ONE         (N)   1.0000
ALFABET     (C)  ABCDEFGHIJKL
CHARS       (C)  ABCDEFGHIJKL NEW STUFF
** TOTAL **       03 VARIABLES USED  00042 BYTES USED
```

RETURN

RETURN

This command is used inside a command file to return control to the command file which called it (or to the keyboard if the user called the command file directly). Encountering an end of file on a command file is equivalent to a RETURN command.

Command files usually have a RETURN command as their last executable line.

See Appendix A for examples.

SAVE

SAVE TO <file>

This command stores all currently defined memory variables to a file. These memory variables may be restored by the RESTORE command.

Examples:

```
. DISPLAY MEMORY
ONE         (N)   1.0000
ALFABET     (C)   ABCDEFGHIJKL
CHARS       (C)   ABCDEFGHIJKL NEW STUFF
** TOTAL **        03 VARIABLES USED  00042 BYTES USED

. SAVE TO MEMFILE

. RELEASE ALL

. DISPLAY MEMORY
** TOTAL **        00 VARIABLES USED  00000 BYTE USED

. RESTORE FROM MEMFILE

. DISPLAY MEMORY
ONE         (N)   1.0000
ALFABET     (C)   ABCDEFGHIJKL
CHARS       (C)   ABCDEFGHIJKL NEW STUFF
** TOTAL **        03 VARIABLES USED  00042 BYTES USED
```

SELECT

```
SELECT [PRIMARY  ]
       [SECONDARY]
```

This command causes dBASE to select one of the two possible database areas for future operations. This permits the dBASE user to do operations on two databases at a time, such as using the data from one database to update the data in another database ,or comparing the data in two databases, or any of a number of other multi-database operations.

When dBASE is initiated, the PRIMARY area is active. PRIMARY will stay active until a SELECT SECONDARY instruction is given. The secondary area will then be active until a SELECT PRIMARY command is encountered. A different database may be USE'ed in each of the areas. This permits the (nearly) concurrent usage of two databases at once. There is no effect if a SELECT SECONDARY is entered when the secondary area is already selected or vice versa with the primary area.

When both database areas have databases in USE, field variables can be extracted from either area. That is to say, any expression can use variables from either database region. If the field names in both regions are the same for a desired variable, then the variable can be prefixed with a "P." or "S." to denote which database it is to come from.

dBASE commands that cause movement of the database (i.e. GOTO, SKIP, REPORT, SORT, COPY, LIST, DISPLAY (for a scope of more than one record), and others) affect only the currently selected database. The SET LINKAGE ON command will allow all sequential commands (those that have a <scope> parameter) perform positioning on both the secondary and the primary databases. (See the SET command). The REPLACE command will only affect variables in the currently selected database. The DISPLAY STRUCTURE command will display the structure of the currently selected database only.

Examples:

```
. USE SHOPLIST
```

```
. LIST
00001  Beans                    5    0.75
00002  Bread loaves             2    1.06
00003  T-Bone steak             4    4.33
00004  Paper plates             1    0.94
00005  Plastic forks            5    0.42
00006  Lettuce                  2    0.53
00007  Bleu cheese              1    1.96
00008  Milk                     2    1.30
00009  Charcoal                 2    0.75

. NOTE NOW OPEN ANOTHER DATABASE IN THE SECONDARY AREA

. SELECT SECONDARY

. USE SHOPCOST

. LIST
00001  800104      31.38
00002  800111      45.69
00003  800118      51.18
00004  800124      48.19
00005  800201      55.82
00006  800209      12.04
00007  800229      12.04

. SELECT PRIMARY

. SUM COST
 12.04

. SELECT SECONDARY

. APPEND

RECORD 00008

DATE       : 800303
AMOUNT     : 12.04

RECORD 00009

DATE       : (cr)

. SUM AMOUNT
 268.38

. NOTE EITHER DATABASE'S VARIABLES CAN BE ACCESSED

. DISP OFF COST,AMOUNT,ITEM,DATE
     0.75  12.04  Charcoal               800303

. NOTE THE SAME DATABASE CAN BE USED IN BOTH AREAS

. USE SHOPLIST
. NOTE BUT ONE MUST BE CAREFUL SINCE THE VARIABLE NAMES ARE IDENTICAL
. NOTE IN BOTH DATABASES
```

SET
---

a. SET <parm1> [ON]
               [OFF]
b. SET <parm2> TO <opt>

This command changes the configuration of dBASE. SET has two forms. Form a allows those parameters that are "toggles" to be set on or off; form b allows those parameters that need one of the different strings described below to have its default reset.

Form a parameters and defaults:

<parm1>	action	meaning
1. ECHO	ON	all commands which come from a command file are echoed on the screen.
	<u>OFF</u>	There is no echo.
2. STEP	ON	dBASE halts after the completion of each command and waits for the user to decide either to go to the next command, quit (escape) from the command file, or enter a command from the keyboard. (STEP is used for debugging command files).
	<u>OFF</u>	Normal operations are resumed.
3. TALK	<u>ON</u>	The results from commands are displayed on the screen.
	OFF	There is no display shown.
4. PRINT	ON	Output is echoed to printer.
	<u>OFF</u>	The echo is turned off.
5. CONSOLE	<u>ON</u>	Output is echoed to the screen.
	OFF	Output to the screen is turned off.

Note: the default values are underlined.

6. ALTERNATE	ON	Output is echoed to a disk file.
	<u>OFF</u>	The echo to the file is turned off.

7.	SCREEN	ON	Full-screen operations are turned on for APPEND, INSERT, EDIT, and CREATE
		OFF	Full-screen operations are turned off.
8.	LINKAGE	ON	Makes all sequential commands (LIST, REPORT, SUM, i. e. commands that have a <scope> parameter) perform positioning on both the PRIMARY and SECONDARY databases.
		OFF	Makes PRIMARY and SECONDARY databases independant.
9.	COLON	ON	Bounds GET data items with colons in @ commands.
		OFF	Removes colons.
10.	BELL	ON	Bell rings whenever illegal data is entered or data boundaries are crossed.
		OFF	Bell is turned off.
11.	ESCAPE	ON	An escape character (1B Hex) aborts execution of command files.
		OFF	There is no escape.
12.	EXACT	ON	Requires that character strings match completely (except for trailing blanks) in expressions and the FIND command.
		OFF	Matches will be made on the basis of the length of the second string, e.g. "ABCDEF" = "ABC" is true.

13.	INTENSITY	ON	Full-screen operations will use dual intensity screen characters (normal and inverse video on some terminals)
		OFF	Dual intensity will not be used.
14.	DEBUG	ON	Output from the ECHO and STEP commands will be sent to the printer so that full-screen commands may be checked out without the screen becoming cluttered.
		OFF	No extra output on the printer.
15.	CARRY	ON	Data from the previous record will be carried-over when APPENDing records in the full-screen mode.
		OFF	No carrying will be done.
16.	CONFIRM	ON	dBASE will not skip to next field in full-screen editing until a control key (like return) is typed.
		OFF	dBASE will skip to next field anytime too many characters are entered.
17.	EJECT	ON	REPORT command will eject a page before beginning a new report.
		OFF	The page eject will be suppressed.
18.	RAW	ON	Places spaces between fields when the DISPLAY and LIST commands are used without the fields list.
		OFF	Spaces are left off.
19.	SCREEN	ON	Uses full-screen for EDIT, APPEND, INSERT and CREATE commands.
		OFF	Turns full-screen capabilities off.

Form b parameters and their formats:

1. SET HEADING TO <string>

This form of the SET command saves the <string> internally and prints the string as part of the report header line. The <string> can be up to 60 characters long. (See REPORT for an example.)

```
2. SET FORMAT TO [SCREEN]
                 [PRINT]
                 [<format file>]
```

The first two forms of this SET parameter determine where the output of "@" commands will go. The last form determines where @ commands are READ from. (See the "@" and READ commands.)

3. SET DEFAULT TO <drive>

This SET commands makes the specified disk drive into the default drive. dBASE will assume that inexplicit file names are on this disk drive. This allows command files to be written in such a way (conveniently) that referenced files may be on any drive in the system. This can also be done with &-macros for further generality in disk drive assignment. In the interactive mode of dBASE, this SET command permits implicit file names.

wnen a default drive has been set, ALL inexplicit filenames are set to the dBASE default. This includes form files, command files, memory files, format files, index files, text files as well as database files.

The parameter <drive> may or may not have the colon (:) attached, that is, both "B" and "B:" are acceptable forms of specifing which drive is wanted,

NOTE: This SET command does not affect the CP/M default drive in any way. The dBASE initial default drive is the same as the CP/M default drive, the SET DEFAULT redefines dBASE's internal default only while within dBASE.

Example:

```
. SET DEFAULT TO B:

. USE DATEVSYR      (dBASE will access the 'B' drive for
                        this database)
```

4. SET ALTERNATE TO [<file>]

This form of the SET ALTERNATE command is part of a two step process to write everything that is normally written onto the screen, onto a disk file as well. This includes output that dBASE generates as well as all inputs typed onto the console. This form identifies and opens the receiving disk file. If the <file> existed on the disk prior to this command, it will be overwritten. A subsequent SET ALTERNATE ON begins the echo process.

Example:

```
SET ALTERNATE TO B:PRINTFLE
SET ALTERNATE ON
 .
 .
 .
any commands
 .
 .
 .
SET ALTERNATE TO anyfile
```

Everything which appears on the screen or printer will be copied onto (in this example) B:PRINTFLE.TXT, which can be word processed, printed, or saved.

5. SET DATE TO mm/dd/yy

The system date can be set or reset at any time with this command. It however does not perform date/calendar validation like the date request when dBASE is first started.

```
SET DATE TO 12,10,76
```

6. SET INDEX TO <index file> [, <index file>, ... <index file>]

SET INDEX TO identifies and sets up as many as seven index files to be used for future operations. If an index file is currently in USE when this command is issued then the old index file is closed and the new one established.

Note: when the new index is set up, the database is left positioned where it was, but, the index does not point anywhere. A FIND command or GOTO must be issued to set the index pointer, before any commands that have a next clause are issued.

The first index file named is considered as the Master Index. All FINDs use only this index and the database will be in the Master Index order (when skipping).

A "SET INDEX TO" command (with no index files) will release all indexes and the database will be a sequential file.

7. SET MARGIN TO n

This form of the SET command allows the user to control the left margin when a report is printed. All lines to be printed will be offset by n spaces. The n parameter must be a literal number in the range 1 to 254.

SKIP

```
SKIP [+][<exp>]
     [-]
```

This command causes the current record pointer to be advanced or backed up relative to its current location.

Example:

```
. USE INVNTRY1

. LIST
00001  136928  13  1673 ADJ. WRENCH      7.13 189     9   0    9.98
00002  221679   9  1673 SM. HAND SAW     5.17 173     4   1    7.98
00003  234561   0    96 PLASTIC ROD      2.18 27    112  53    4.75
00004  556178   2   873 ADJ. PULLEY     22.19 117     3   0   28.50
00005  723756  73    27 ELECT.BOX       19.56 354     6   1   29.66
00006  745336  13    27 FUSE BLOCK      12.65 63      7   2   15.95
00007  812763   2  1673 GLOBE            5.88 112     5   2    7.49
00008  876512   2   873 WIRE MESH        3.18 45      7   3    4.25
00009  915332   2  1673 FILE             1.32 97      7   3    1.98
00010  973328   0    27 CAN COVER        0.73 21     17   5    0.99

. 5

. SKIP -2
RECORD: 00003

. SKIP
RECORD: 00004

. SKIP 3
RECORD: 00007
```

SORT

```
SORT ON <field> TO <file> [ASCENDING ]
                          [DESCENDING]
```

This command allows the user to sort data files to another file which is different from the original file. The file in USE is sorted on one of the data fields and may be sorted into ascending or descending order. Notice that the USE file remains in USE and is unaltered.

While the SORT command allows only one key, a database may be sorted on several keys by cascading sorts: sort on the most minor key first and progress toward the major key. dBASE will only disturb the order of records when necessary. The collating sequence for character fields is the ASCII code. ASCENDING is assumed if neither ASCENDING or DESCENDING is specified.

The sort uses the ASCII collating sequence. This means that the string 'SMITH' is "smaller" than 'Smith' (the expression "'SMITH' < 'Smith'" would be TRUE).

The INDEX command is contrasted with the SORT command in this way: INDEX, when done, performs nearly all of SORTs dutys. Also, INDEX generally allows greater freedom and greater speed than SORT.

```
. USE SHOPLIST

. LIST
00001  BEANS #303 CAN         5      0.75
00002  BREAD LOAVES           2      0.97
00003  T-BONE STEAKS          4      3.94
00004  PAPER PLATES           1      0.86
00005  PLASTIC FORKS          5      0.42
00006  LETTUCE                2      0.53
00007  BLEU CHEESE            1      1.96
00008  MILK (1/2 GAL)         2      1.30
00009  CHARCOAL, 5# BAGS      2      0.75

. SORT ON ITEM TO SORTFILE
SORT COMPLETE

. USE SORTFILE
```

```
. LIST
00001  BEANS #303 CAN         5      0.75
00002  BLEU CHEESE            1      1.96
00003  BREAD LOAVES           2      0.97
00004  CHARCOAL, 5# BAGS      2      0.75
00005  LETTUCE                2      0.53
00006  MILK (1/2 GAL)         2      1.30
00007  PAPER PLATES           1      0.86
00008  PLASTIC FORKS          5      0.42
00009  T-BONE STEAKS          4      3.94
```

STORE

```
STORE <exp> TO <memvar>
```

This command computes the value of an expression and stores the value into a memory variable. If the memory variable did not exist before this command was issued then dBASE will create the memory variable automatically.

Note that STORE will alter only memory variables. Use the REPLACE command to change database field variables.

```
. RELEASE ALL

. STORE 1 TO ONE
 1

. STORE 'ABCDEFGHIJKL' TO ALFABET
ABCDEFGHIJKL

. STORE ALFABET+' NEW STUFF' TO CHARS
ABCDEFGHIJKL NEW STUFF

. STORE ONE*1.0000 TO ONE
 1.0000

. DISPLAY MEMORY
EOF       (L)  .T.
ONE       (N)   1.0000
ALFABET   (C)  ABCDEFGHIJKL
CHARS     (C)  ABCDEFGHIJKL NEW STUFF
** TOTAL **     04 VARIABLES USED  00042 BYTES USED
```

SUM

```
SUM <field> [,<field>] [TO <memvar list>]
    [<scope>] [FOR <exp>]
```

The SUM command adds numeric expressions involving the USE file according to the <scope> and FOR clauses. Up to 5 expressions may be simultaneously summed. If the TO clause is present, the sums are also stored into memory variables (memory variables will be created if they didn't exist prior to the issuance of the sum command). The default scope of SUM is all non-deleted records.

```
. USE SHOPLIST

. LIST
00001  BEANS #303 CAN          5     0.75
00002  BREAD LOAVES            2     0.97
00003  T-BONE STEAKS           4     3.94
00004  PAPER PLATES            1     0.86
00005  PLASTIC FORKS           5     0.42
00006  LETTUCE                 2     0.53
00007  BLEU CHEESE             1     1.96
00008  MILK (1/2 GAL)          2     1.30
00009  CHARCOAL, 5# BAGS       2     0.75

. SUM COST
 11.48

. SUM COST FOR NO=1
 2.82

. SUM COST,NO
 11.48  24

. SUM COST TO MSUM
 11.48

. ? MSUM
 11.48

. DISPLAY MEMORY
MSUM          (N)    11.48
** TOTAL **          01 VARIABLES USED  00006 BYTES USED

. ? MSUM*1.10
12.6280

. SUM NO*COST,NO,COST,COST/NO
 31.53  24 11.48 5.81
```

TOTAL

TOTAL ON <key> TO <database> [FIELDS <list>] [FOR <expression>]

The TOTAL command is similar to the subtotal capability in the REPORT command except that the subtotals are placed into a database instead of printed. This allows condensation of data by eliminating detail and summarizing.

Note: the USE database must be either presorted by the key or indexed on the key.

If the TO database was defined (if it existed and had a structure), then it's structure will be left intact and used to decide which fields will be totalled arithmetically.

If the TO database did not exist prior to this TOTAL command, then the structure will be constructed using the field names given by the FIELDS phrase. If there is no FIELD phrase then the structure from the USE database will be copied to the TO file.

This command is most selective when the TO database exists and the FIELD phrase is included in the command. In this case, only the numeric fields in the FIELDS are totalled. In any other configuration of this command, all numeric fields are totalled.

TOTAL can also be used to remove duplicate records from a database since a non-numeric field in the FIELDS list is not totalled (naturally) and is not flagged as an error.

Example:

```
. **USE ORDERS INDEX ORDERS**

. **DISPLAY STRU**
STRUCTURE FOR FILE:    ORDERS.DBF
NUMBER OF RECORDS:     00008
DATE OF LAST UPDATE:   00/00/00
PRIMARY USE DATABASE
FLD        NAME       TYPE WIDTH  DEC
001      CUSTOMER      C    020
002      PART:NO       C    005
003      AMOUNT        N    005
** TOTAL **               00031
```

```
. LIST
00003  HARRIS, ARNOLD        11528    44
00007  JUAN, DON             21828     5
00001  SWARTZ, JOE           31415    13
00005  MACK, JAY             31415     3
00008  SALT, CLARA           70296     9
00002  SWARTZ, JOE           76767    13
00006  TERRY, HANS           76767     5
00004  ADAMS, JEAN           89793    12
```

(Imagine that the warehouse needs to know how many of each item to bring out. By totaling on the quantity as long as the part numbers are the same, a database is generated that contains part numbers and the number needed)

(The database CALLS has already been defined)

```
. TOTAL ON PART:NO TO CALLS
00006 RECORDS COPIED

. USE CALLS

. DISP STRU
STRUCTURE FOR FILE:    CALLS.DBF
NUMBER OF RECORDS:     00006
DATE OF LAST UPDATE:   00/00/00
PRIMARY USE DATABASE
FLD        NAME       TYPE WIDTH  DEC
001       PART:NO      C    005
002       AMOUNT       N    005
** TOTAL **               00011

. LIST
00001  11528    44
00002  21828     5
00003  31415    16       (Note: two orders totaled)
00004  70296     9
00005  76767    18       (Note: two other orders totaled)
00006  89793    12
```

UPDATE

```
UPDATE FROM <database> ON <key> [ADD <field list>]
                                [REPLACE <field list>]
```

The UPDATE command revises the USE file by using data from a second database to modify the USE database. Updated items can be summed or replaced in entirety. A record is updated when the criterion is met by the comparison of a field in the USE database with one from the FROM database. These fields are known as the key and are supplied with the ON phrase.

Note: the USE database must be either pre-sorted by the key or indexed on the key. The FROM database must be pre-sorted by the key.

Both databases are 'rewound' and a record is read. If the keys match, the add or replace action takes place as directed. If the key in the USE file is smaller (in sort sequence) than the key in the FROM database, then no action takes place, and the record is skipped and left unchanged. Similarly, if the FROM key is smaller, no updates happen and that record is skipped.

Example:

```
. USE INVUPDAT

. DISPLAY STRUCTURE
STRUCTURE FOR FILE:   INVUPDAT.DBF
NUMBER OF RECORDS:    00003
DATE OF LAST UPDATE:  00/00/00
PRIMARY USE DATABASE
FLD        NAME          TYPE WIDTH    DEC
001      PART:NO          C    005
002      ON:HAND          N    005
003      COST             N    010     002
** TOTAL **                   00021

. LIST
00001  21828     77      35.88
00002  70296      0     250.00
00003  89793      2  134999.00
```

(Notice that the database is sorted on the "key" PART:NO.).

```
. USE INVENTRY INDEX INVENTRY

. DISPLAY STRUCTURE
STRUCTURE FOR FILE:   INVENTRY.DBF
NUMBER OF RECORDS:    00008
DATE OF LAST UPDATE:  00/00/00
PRIMARY USE DATABASE
FLD       NAME         TYPE WIDTH   DEC
001       ITEM          C    020
002       COST          N    010    002
003       PART:NO       C    005
004       ON:HAND       N    005
** TOTAL **                 00041

. DISP ALL
00008  #9 COAL                    22.00 11528   16
00005  SINK, KITCHEN              34.72 21828   77
00001  TIME STITCH                 9.99 24776    1
00002  WIDGET                      1.67 31415   18
00007  RINGS, GOLDEN             200.00 70296    5
00006  TROMBONES                 198.37 76767   76
00004  TANK, SHERMAN          134999.00 89793    5
00003  GADGET, LARGE              16.33 92653    7
```

(Again notice that the database is indexed on the "key" PART:NO.)

```
. UPDATE ON PART:NO FROM INVUPDAT ADD ON:HAND REPLACE COST

. LIST
00008  #9 COAL                    22.00 11528   16
00005  SINK, KITCHEN              35.88 21828  154
00001  TIME STITCH                 9.99 24776    1
00002  WIDGET                      1.67 31415   18
00007  RINGS, GOLDEN             250.00 70296    5
00006  TROMBONES                 198.37 76767   76
00004  TANK, SHERMAN          134999.00 89793    7
00003  GADGET, LARGE              16.33 92653
```

(Note--the two new Sherman tanks were added to the database and the cost of the golden rings and the kitchen sinks were replaced with the new prices.)

USE

```
USE [<database file>]
USE <databasefile> INDEX <index file> [, <index file>, ... <index file>]
```

Example:

```
. USE DATABASE INDEX NAME,CITY,PART:NO,SALESMAN
```

The USE command specifies which (pre-existing) database file is to be the file in USE. If there was a USE file prior to this command, the old file is closed. If a filename is not specified in the command, then the previous USE file is closed.

The second form of USE is to specify a database for operation and an associated index file (which was previously created by the INDEX command or the SET INDEX TO <index file> command) and permits subsequent index operations such as FIND and indexed sequential file access.

Up to seven index files may be USEd with any one database at the same time. The first index file named is considered as the Master Index. All FINDs use only this index and the database will be in the Master Index order (when skipping). All of the named index files will be automatically updated anytime their keys are modified (by APPEND, EDIT, REPLACE, READ, or BROWSE commands).

Examples:

```
. USE EXAMPLE

. USE TRACE INDEX TRACE
```

WAIT

```
WAIT [TO <memvar>]
```

This command causes dBASE to cease operations until any character is entered from the keyboard, the message WAITING is displayed on the screen. If the TO clause is specified, then the single keystroke that releases dBASE from the wait-state will be entered into the memory variable.

The TO option is most useful when only a single character is required to direct the action of a command file process e.g. menu selections. Notice that a carriage return is not necessary to "send" the character as in the ACCEPT and INPUT commands.

If any non-printable character (i.e. RETURN, LINE FEED, or any other control character) is typed as the response to a WAIT TO command, the value of the memory variable is set to a blank.

Example:

```
. RELEASE ALL

. WAIT TO ACTION
WAITING 1

. DISP MEMO
ACTION      (N)    1
** TOTAL **        01 VARIABLES USED  00006 BYTES USED
```

APPENDIX A COMMAND FILE EXAMPLE

The following is one example of how command files may be used in a practical environment. In this example, the command files are used like a program written in a more classical language. Command files can contain groups of commands which perform some smaller function e.g. a series of SORT's.

This example is a simple checkbook balancing and check register maintenance system. It consists of 4 command files: the controlling file, MENU, and three subordinate files, NEWENTR, CANCELS, and BALANCE. This problem solution could be structured in many different ways; here, this example has been structured to show the dBASE commands that deal especially with command files.

The command files were created by a text editor using the type ".CMD" in order to facilitate their usage. The sample run is an actual output of dBASE using the SET ALTERNATE technique. Refer to the SET command for this technique.

In solving any database problem, one should first consider what data fields will be required. For this example, the following fields were selected:

NO	- the check number
TO	- the recipient of the check
AMT	- the dollar amount of the check
CAN	- the cancelled/not-cancelled status of a check
DATE	- the date on which the check was written

dBASE is then entered to CREATE the database structure.

```
. CREATE
FILENAME:CHECKREG
ENTER RECORD STRUCTURE AS FOLLOWS:
 FIELD    NAME,TYPE,WIDTH,DECIMAL PLACES
 001      NO,N,4
 002      TO,C,30
 003      AMT,N,10,2
 004      CAN,L
 005      DATE,C,10
 006      (cr)
INPUT NOW?N
```

A text editor is then executed and the following command file sources are entered:

First the MENU command file;

```
NOTE -  Example dBASE Command file program
*
*
SET TALK off
USE CHECKREG
DO WHILE T
   ?
   ?
   ?
   ? '         Checkbook Balancer Menu'
   ?
   ?
   ? '         0 - EXIT'
   ? '         1 - Enter New Checks'
   ? '         2 - Enter Cancelled Checks'
   ? '         3 - Balance'
   ?
   ? ' enter desired action'
   WAIT TO ACTION
   IF ACTION='0'
      SET TALK on
      CANCEL
   ENDIF
   IF ACTION='1'
      DO NEWENTR
   ENDIF
   IF ACTION='2'
      DO CANCELS
   ENDIF
   IF ACTION='3'
      DO BALANCE
   ENDIF
ENDDO
RETURN
```

Second the NEWENTR command file

```
NOTE - NEWENTR Command File to Enter New Checks
*
REMARK Enter Check Number of 0 to Exit
DO WHILE T
   ?
   ?
   INPUT "Enter Check Number       " to C:NO
   IF C:NO=0
      RETURN
   ENDIF
   ? '
   ACCEPT "Paid to Order of        " to C:TO
   INPUT "Amount of Check          " to C:AMT
   ACCEPT "Date of Check           " to C:DAT
   ?
   INPUT "Are all fields correct ? " to GO:NOGO
   IF .NOT.GO:NOGO
      LOOP
   ENDIF
   APPEND BLANK
   REPLACE NO with C:NO, TO with C:TO, AMT with C:AMT, DATE ;
     with C:DAT, CAN with F
ENDDO
```

Third the CANCELS command file

```
NOTE - CANCELS Command file to enter cancelled checks
*
REMARK  Enter Check Number of 0 to Exit
DO WHILE T
   ?
   INPUT "Enter Cancelled Check no " to C:CAN
   IF C:CAN=0
      RETURN
   ENDIF
   GO TOP
   LOCATE for C:CAN=NO
   REPLACE CAN with T
ENDDO
```

Last the BALANCE command file

```
NOTE - BALANCE Command File to Balance Checkbook
*
SUM AMT to OUTSTAND for .NOT.CAN
?
?
DISPLAY off 'Total Outstanding Checks = $',OUTSTAND
?
REMARK  Enter Outstanding Deposits, Enter 0 to Proceed
STORE T to ACTIVE
STORE 1 to COUNT
STORE 0 to T:OUT
DO WHILE ACTIVE
   STORE STR(COUNT,3) to I
   INPUT 'Enter Amount of Outstanding Deposit &I ' to D:OUT
   IF D:OUT=0
      STORE F to ACTIVE
   ELSE
      STORE D:OUT+T:OUT to T:OUT
      STORE COUNT+1 to COUNT
   ENDIF
ENDDO
DISPLAY OFF COUNT-1,' Total Outstanding Deposits   Total = $',T:OUT
?
INPUT "Enter Ending Balance" to BEGIN
DISPLAY OFF 'Current Balance = $',BEGIN+T:OUT-OUTSTAND
WAIT
RETURN
```

A sample run of these command files follows:

```
. DO MENU


        Checkbook Balancer Menu


        0 - EXIT
        1 - Enter New Checks
        2 - Enter Cancelled Checks
        3 - Balance

 enter desired action
WAITING 1
 Enter Check Number of 0 to Exit


Enter Check Number        :1000

Paid to Order of          :ACME Rentals
Amount of Check           :123.45
Date of Check             :10 Jun 79

Are all fields correct ? :y


Enter Check Number        :1001

Paid to Order of          :Mag Publishing Co.
Amount of Check           :79.88
Date of Check             :12 Jun 79

Are all fields correct ? :y


Enter Check Number        :1002

Paid to Order of          :Radon Inert Gases
Amount of Check           :86.86
Date of Check             :13 Jun 79

Are all fields correct ? :y
```

```
Enter Check Number        :1003

Paid to Order of          :Neuron Comm. Inc.
Amount of Check           :723.31
Date of Check             :14 Jun 79

Are all fields correct ? :y


Enter Check Number        :1004

Paid to Order of          :Crankshaft Auto
Amount of Check           :2753.47
Date of Check             :19 Jun 79

Are all fields correct ? :y


Enter Check Number        :0



        Checkbook Balancer Menu


        0 - EXIT
        1 - Enter New Checks
        2 - Enter Cancelled Checks
        3 - Balance

 enter desired action
WAITING 2
 Enter Check Number of 0 to Exit

Enter Cancelled Check no :1001

Enter Cancelled Check no :1003

Enter Cancelled Check no :0


        Checkbook Balancer Menu


        0 - EXIT
        1 - Enter New Checks
        2 - Enter Cancelled Checks
        3 - Balance

 enter desired action
WAITING 3
Total Outstanding Checks = $ 2963.78

 Enter Outstanding Deposits, Enter 0 to Proceed
Enter Amount of Outstanding Deposit    1 :1234.56
Enter Amount of Outstanding Deposit    2 :.03
Enter Amount of Outstanding Deposit    3 :333.44
Enter Amount of Outstanding Deposit    4 : 0
  3 Total Outstanding Deposits   Total = $ 1568.03

Enter Ending Balance:1445.89
Current Balance = $ 50.14
WAITING


         Checkbook Balancer Menu


         0 - EXIT
         1 - Enter New Checks
         2 - Enter Cancelled Checks
         3 - Balance

 enter desired action
WAITING 0
DO CANCELLED
```

At this point, the user could easily do direct dBASE commands to interrogate, modify, or report on the database file. For instance the commands:

```
DISPLAY DATE,AMOUNT for NO=1003
```

or

```
SUM AMT for DATE>'01 Jun'
```

or any other dBASE commands could be issued to provide information as needed to accommodate unforeseen circumstances in the course of managing a checkbook.

APPENDIX B   LIST OF COMMANDS

```
? <exp> [,<exp>]
@ <coordinates> [SAY <exp> [USING '<picture>']] [GET
         <variable> [PICTURE '<picture>']]
ACCEPT ["<cstring>"] TO <memvar>
APPEND [FROM <file> [SDF] [DELIMITED] [FOR <exp>]]
         or [BLANK]
BROWSE
CANCEL
CHANGE FIELD <list> [<scope>] [FOR <exp>]
CLEAR [GETS]
CONTINUE
COPY TO <file> [<scope>] [FIELD <list>] [FOR <exp>]
         [SDF] [DELIMITED [WITH <delimiter>]] or [STRUCTURE]
COUNT [<scope>] [FOR <exp>] [TO <memvar>]
CREATE [<filename>]
DELETE [<scope>] [FOR <exp>]
DELETE FILE <file>
DISPLAY [<scope>] [FOR <exp>] [<exp list>] [OFF]
DISPLAY STRUCTURE
DISPLAY MEMORY
DISPLAY FILES [ON <disk drive>] [LIKE <skeleton>]
DO <file>
DO WHILE <exp>
EDIT
EJECT
ELSE
ENDDO
ENDIF
ERASE
FIND <key>
GO or GOTO [RECORD], or [TOP], or [BOTTOM], <n>
IF <exp>
INDEX ON <char string expression> TO <index file name>
INPUT ["<cstring>"] TO <memvar>
INSERT [BEFORE], or [BLANK]
JOIN TO <file> FOR <expression> [FIELDS <field list>]
LIST
LOCATE [<scope>] [FOR <exp>]
LOOP
MODIFY STRUCTURE
MODIFY COMMAND <command file>
NOTE or *
PACK
QUIT [TO <list of CP/M level commands or .COM files>]
READ
RECALL [<scope>] [FOR <exp>]

RELEASE [<memvar list>], or [ALL]
REMARK
RENAME <current file name> TO <new file name>
REPLACE [<scope>] <field> WITH <exp> [AND <field> WITH <exp>]
REPORT [<scope>] [FORM <form file>] [TO PRINT] [FOR <exp>]
RESET
```

```
RESTORE
RETURN
SAVE TO <file>
SELECT [PRIMARY or SECONDARY]
SET <parm> [ON], or [OFF]
SET ALTERNATE TO <file>
SET DEFAULT TO <drive>
SET DATE TO <string>
SET FORMAT TO <format file name>
SET HEADING TO <string>
SET INDEX TO <index file>
SET MARGIN TO <n>
SKIP <+/-> [<n>]
SORT ON <field> TO <file> [ASCENDING], or [DESCENDING]
STORE <exp> TO <memvar>
SUM <field> [<scope>] [TO <memvar list>] [FOR <exp>]
TOTAL TO <file> ON <key variable> [FIELDS <field list>]
UPDATE FROM <file> ON <key variable> [ADD <field list>]
     [REPLACE <field list>]
USE <file> [INDEX <index file name>]
WAIT [TO <memvar>]
```

FUNCTIONS:

@(<string1>,<string2>)	AT function
*	deleted record func
#	record number func
!(<char string>)	upper case function
$(<char string>,<start>,<length>)	substring function
<string1>$<string2>	substring search
CHR(<numeric expression>)	numeric to ASCII
DATE()	system date func
EOF	end-of-file func
FILE(<file>)	existance func
INT(<numeric expression>)	integer function
LEN(<char string>)	length function
STR(<numeric expression>,<width>[,<decimals>])	string func
VAL(<char string>)	value function
TRIM(<char string>)	trims strings
TYPE(<exp>)	supplies data type

APPENDIX C LIMITATIONS AND CONSTRAINTS

number of fields per record	32 max
number of characters per record	1000 max
number of records per database	65535 max
number of characters per character string	254 max
accuracy of numeric fields	10 digits
largest number	1.8 x 10**63 approx
smallest number	1.0 x 10**-63 approx
number of memory variables	64 max
number of characters per command line	254 max
number of expressions in SUM command	5 max
number of characters in REPORT header	254 max
number of characters in index key	100 max
number of pending GETS	64 max
number of files open at one time	16 max

APPENDIX D ERROR MESSAGES

BAD DECIMAL WIDTH FIELD

BAD FILE NAME
 Syntax error in filename.

BAD NAME FIELD

BAD TYPE FIELD
 Must be C, N, or L.

BAD WIDTH FIELD

CANNOT INSERT - THERE ARE NO RECORDS IN DATABASE FILE
 Use the APPEND command instead.

CANNOT OPEN FILE
 Internal error, contact dealer for support.

COMMAND FILE CANNOT BE FOUND
 Check spelling.

DATA ITEM NOT FOUND

DATABASE IN USE IS NOT INDEXED
 FIND is only permitted on indexed databases.

DIRECTORY IS FULL
 The CP/M disk directory cannot hold anymore files.

DISK IS FULL

END OF FILE FOUND UNEXPECTEDLY
 The database in USE is not in the correct format. If all records are correct and present, then PACK and re-INDEX the database

"FIELD" PHRASE NOT FOUND

FILE ALREADY EXISTS

FILE DOES NOT EXIST

FILE IS CURRENTLY OPEN
 Type a USE or CLEAR command to close the file.

FORMAT FILE CANNOT BE OPENED

FORMAT FILE HAS NOT BEEN SET

ILLEGAL DATA TYPE

ILLEGAL GOTO VALUE

ILLEGAL VARIABLE NAME
Only alphanumerics and colons are allowed in variable and field names.

INDEX DOES NOT MATCH DATABASE
dBASE cannot match the key with the database. Try another index file.

INDEX FILE CANNOT BE OPENED
Check spelling or INDEX the database.

JOIN ATTEMPTED TO GENERATE MORE THAN 65,534 RECORDS
The FOR clause allows too many joined output records, make it more stringent.

KEYS ARE NOT THE SAME LENGTH

MACRO IS NOT A CHARACTER STRING
&macros must be character strings.

MORE THAN 5 FIELDS TO SUM

NESTING LIMIT VIOLATION EXCEEDED

NO EXPRESSION TO SUM

NO "FOR" PHRASE

NO "FROM" PHRASE

NO FIND
More a diagnostic type message than an error message. dBASE couldn't find the key.

NON-NUMERIC EXPRESSION

NONEXISTENT FILE

"ON" PHRASE NOT FOUND

OUT OF MEMORY FOR MEMORY VARIABLES
Reduce the number or size of memory variables.

RECORD LENGTH EXCEEDS MAXIMUM SIZE (OF 1000)

RECORD NOT IN INDEX
Index file was not updated after a record was added. Reindex.

RECORD OUT OF RANGE
Record number greater than number of records in database. The Record doesn't exist.

SORTER INTERNAL ERROR, NOTIFY SCDP
Internal error, contact dealer for support.

SOURCE AND DESTINATION DATA TYPES ARE DIFFERENT

*** SYNTAX ERROR ***

SYNTAX ERROR IN FORMAT SPECIFICATION

SYNTAX ERROR, RE-ENTER

"TO" PHRASE NOT FOUND

TOO MANY CHARACTERS

TOO MANY FILES ARE OPEN
There is a maximum of 16 files allowed to be open at one time.

TOO MANY MEMORY VARIABLES
There is a maximum of 64 memory variables.

TOO MANY RETURNS ENCOUNTERED
Probably an error in the structure of a command file.

"WITH" PHRASE NOT FOUND

UNASSIGNED FILE NUMBER
Internal error, contact dealer for support.

*** UNKNOWN COMMAND

VARIABLE CANNOT BE FOUND
Need to create the variable, or check the spelling.

INDEX

NOTES

Additional user data about dBASE II operation not yet included in the Manual.

1. The 0th line on the screen is now reserved for special purposes. Therefore, do not issue a format command like '@ 0,<y> SAY <exp>'.

2. The REPORT command has a limit of 24 data fields.

3. Under MP/M the QUIT TO <filename> will not operate.

4. PACK will not reduce amount of disk space reserved for that file by CP/M. To recover the space, use a COPY TO <filename> and then delete the source file. This is a limitation of the CP/M operating system not of dBASE II.

5. DO NOT RENAME a file in USE. Generally it is not even a good practice to RENAME a file while under command program control.

6. The proper syntax for the COPY STRUCTURE command is:
   ```
   USE <file>
   COPY STRUCTURE TO <newfile>
   ```
   the 'STRUCTURE' option should immediately follow the verb 'COPY'.

7. When calling a dBASE data file into USE, do not use the '.DBF' extension. dBASE adds this extension automatically.