dBASE II

Assembly Language

Relational Database Management System

REFERENCE MANUAL

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 2.2

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.

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.

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.

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 opposed 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()
08/15/81
. STORE DATE() TO MEMVAR
08/15/81
. 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
 18
. ? 4+(2*3)
 10
```

An example of use of arithmetic parentheses used for grouping 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
. STORE 'ABCDE   ' TO A
ABCDE
. STORE '1234  67' TO B
1234  67
. ? A-B
ABCDE1234  67
```

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

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.

but only for the purpose of that operation
in memory it remains a char. string.

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.