# Language Reference

# Language Reference

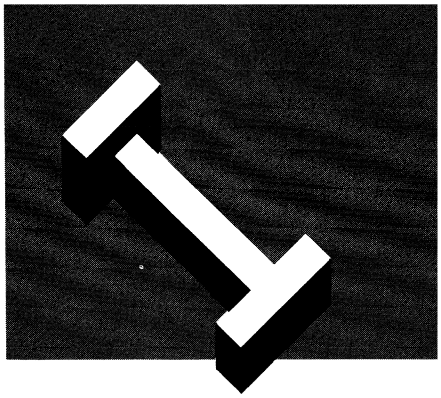
**9/88**

# Contents

# Commands

## System Memory Variables

# Introduction



*Language Reference* is an encyclopedia of dBASE IV™ commands, functions, system memory variables, and system configuration. This manual is for you, if you have completed *Learning dBASE IV* or *Using the Menu System*. If you are already familiar with dBASE® programming, or database management, you will also find this manual a useful reference tool.

If you are an intermediate-to-advanced user and purchased dBASE IV Developer's Edition, you may also wish to refer to *Programming with dBASE IV* and the *Sample Programming Code* booklet.

This manual is organized into six chapters, seven appendixes, and an index.

## Chapter 1: Essentials

This contains a discussion of the dBASE IV basics: using commands, SET commands, functions, and system memory variables. It covers the dBASE IV language components and how to use them to build a command line.

## Chapter 2: Commands

This chapter provides an alphabetical listing of the commands you can use in the dBASE IV language. Each entry contains a basic syntax paradigm and notes on how to use the command. Many commands also contain tips, examples, and cross references to other commands and functions.

The database and index files used in the examples are listed in Appendix C, "Sample Files." Also in Appendix C is a complete listing of Menus.prg, a program from which several examples in this chapter are taken.

## Chapter 3: SET Commands

These are a subset of dBASE IV commands, listed in alphabetical order. SET commands allow certain settings that control how dBASE presents information on the screen or printer, or that establish an environment affecting the way other commands and functions operate.

# Chapter 4: Functions

Chapter 4 contains the same type of information for the functions as Chapters 2 and 3 contain for commands. Each function is listed in alphabetical order and thoroughly described. The description includes a definition, syntax, usage, examples, and any tips that make it easier for you to use.

# Chapter 5: System Memory Variables

System memory variables are a class of memory variables that dBASE IV reserves to hold information about printing and the format of printed material. This chapter contains information about tailoring your printing tasks by changing the values of these variables.

# Chapter 6: Customizing dBASE IV

Certain screen, printer, and memory parameters are initialized when you start up dBASE IV. This chapter provides information on these parameters, and explains how to change the environment that dBASE initializes in memory upon loading.

# Appendixes

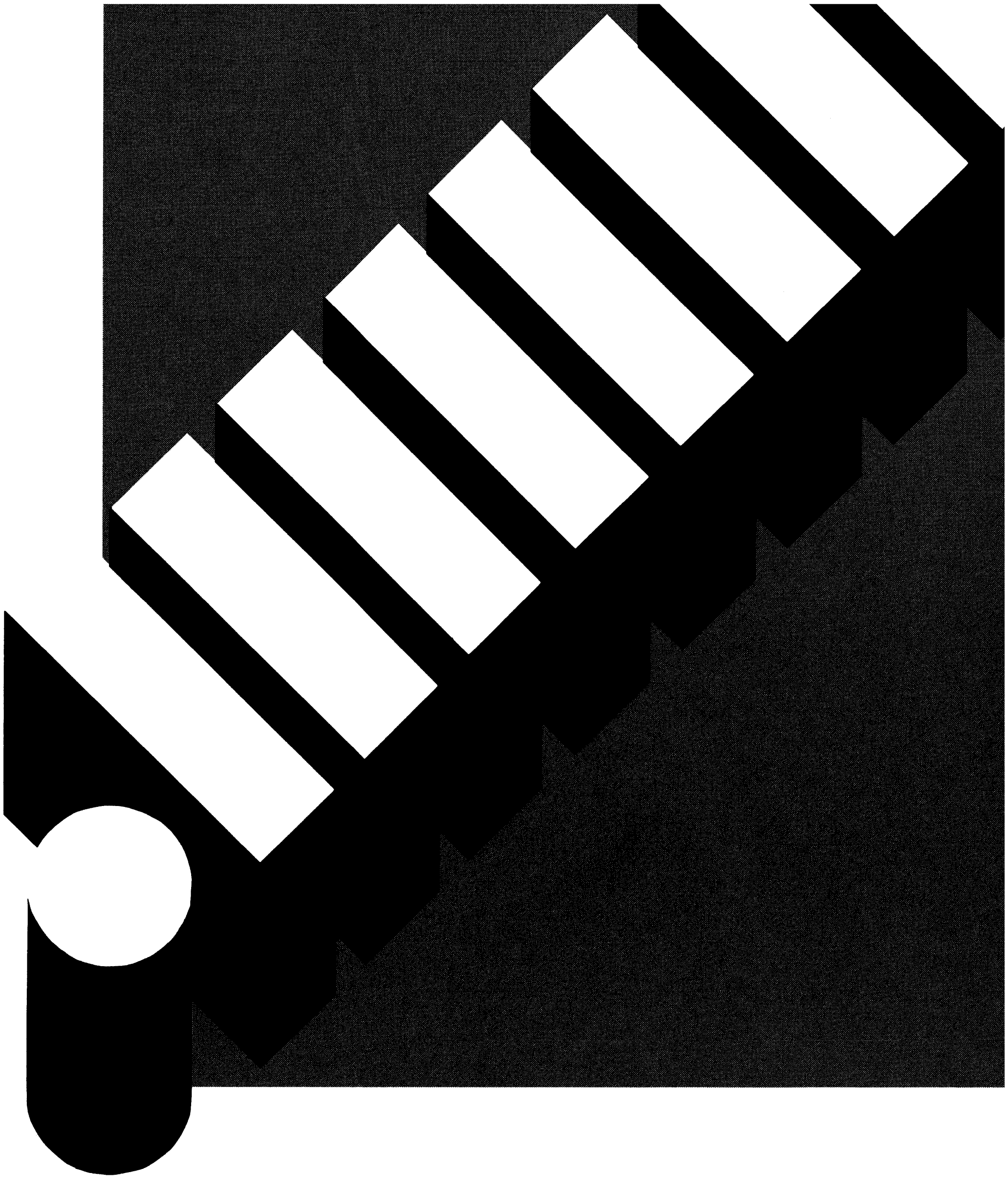
These are as follows:

- A list of error messages and applicable code numbers, along with explanations of possible causes of the messages
- dBASE IV technical specifications
- The sample files that are used for all examples in this manual
- A summary of the types of files that dBASE IV uses and creates, and the file extensions that are written to disk when these files are saved
- Notes on the structure of a database (.dbf) file
- A list of available printer drivers, and the type styles each supports
- The ASCII chart, which provides hexadecimal and decimal equivalents for each character

# Index

This is a convenient cross-reference to all the concepts, commands, functions, and variables discussed in the *Language Reference* manual.

# Essentials



i 1 2 3 4 5 6 A B C

D E F G In

# Essentials

The Control Center lets you easily use commands, SET commands, functions, and system memory variables without knowing the details of their operation. You may also build your own *command line* at the *dot prompt* or within a *program,* if you are familiar with the dBASE IV language components.

## About This Chapter

In this chapter you will find the basic essentials needed to construct a dBASE IV command line. A command line is a complete instruction that directs the dBASE IV processor to act on data items, and may contain a *command* (or *SET command*), *functions,* and *system memory variables.* You can find further information on each of these, respectively, in Chapters 2, 3, 4, and 5.

You may enter a command line at the *dot prompt,* or as part of a *program.*

### dBASE IV Language Components

Each language component has a role in the command line:

- Commands are verbs that direct the dBASE processor to perform a certain action. Although a command line may optionally contain another language component, it must contain one (and only one) command. Sometimes the command verb is implied in the command line, as with the STORE command. The command lines:

  ```
  . STORE 5 TO x
  ```

  and

  ```
  . x = 5
  ```

  are the same, but the STORE command is implied in the second form.

  Because each command line must contain a command verb, the term *command line* is often simply referred to as the *command.*

- SET commands are a subset of commands that usually set up the environment in which the processor acts. For example, SET COLOR sets the colors on the screen and affects other commands that output to the screen.
- Functions work in three ways:
  1. Some functions are adjectives or adverbs that modify a data item. For example, LOWER() converts upper-case letters to lower case.
  2. Other functions hold a dBASE IV value or condition that you can query. The function EOF() will be set to true (.T.) when you are at the end of a database file, and the command:

```
. ? EOF()
```

  returns

```
.T.
```

  3. Some functions are formulas that are evaluated with respect to the input parameters. For example, SQRT(x) returns the square root of the input parameter *x*.
- System memory variables are settings that control the appearance of printed and screen output. While commands, SET commands, and functions were language components in earlier versions of dBASE II®, dBASE III®, and dBASE III PLUS™, system memory variables are new to dBASE IV.

  System memory variables are like SET commands in that they usually control system parameters rather than act on data items. They are like functions in that you can query the values they contain.

In this manual, commands are printed in upper case, such as LIST. SET commands are also upper case, and always begin with the word SET, such as SET DEVICE. Functions are also printed in upper case, but end with parentheses, such as FOUND(). System memory variables are lower case, and begin with an underscore, such as _padvance. These conventions help you distinguish among the terms; for example, CHANGE() is a function, but CHANGE is a command.

## Dot Prompt Interface

dBASE IV's interactive mode, which allows you to enter a command line and get an immediate response, is signaled by a dot on the screen and is therefore known as the *dot prompt*. Using the dot prompt may give you more speed and flexibility than if you work only from the Control Center.

You issue a command in dBASE IV by typing it at the dot prompt and pressing ↵. Each line you enter may be up to 254 characters long.

If you are typing a long command line at the dot prompt, press **Ctrl-Home** to open an *editing window* on the screen. When typing commands in the editing window, you have access to all the features of dBASE IV's text editor, and you can see the entire instruction without scrolling back and forth from the beginning of the command line to the end. In an editing window, the total command line may contain a maximum of 1,024 characters.

Commands and certain keywords may be abbreviated to the first four characters (except SQL commands). You can abbreviate REPORT FORM to REPO FORM and MODIFY COMMAND to MODI COMM, for example, but you must use the entire SQL command.

You may enter command lines in upper case, lower case, or a combination of the two. You may also include any number of blank spaces between the words of a command line. Each blank space, however, counts as one character of the 1024-character maximum per command line.

### Re-entering Commands

The dot prompt has a memory buffer called *history* that automatically stores commands as you enter them. This lets you go back and edit or run a previous command. When you installed dBASE IV, a default of 20 commands was set for the history buffer. You can use SET HISTORY to change the default number of stored commands from 20 to a number from 0 to 16,000. You can also reconfigure the default by changing your Config.db file (see Chapter 6, "Customizing dBASE IV").

Press ↑ at the dot prompt to display commands previously stored in the history buffer. The commands appear one at a time in reverse order. ↓ moves the cursor back down the list of commands. You can run the stored command when it is at the dot prompt by pressing ↵. Use DISPLAY HISTORY and LIST HISTORY to view more than one command at a time.

## Filenames and Aliases

Appendix D lists the types of files dBASE IV creates and uses. You can use aliases to access and relate the information from several different database files.

### Filenames

A filename may be the actual name of a file as it is written on disk, or an *indirect reference* to the filename. Indirect file references are discussed later in this section. If you enter the name of a file, you may also use the drive specifier and full path to indicate where the file can be found.

dBASE IV accepts any valid DOS filename. When you write a file to disk, dBASE IV assigns an extension to the file that indicates the type of information it contains. For example, the CREATE command will write a file with a .dbf extension, which indicates to other dBASE commands that the file contains data records. Appendix D provides a complete listing of the dBASE IV file extensions.

**NOTE**
*Do not use a DOS device name as a filename. Check your DOS manual for DOS device names.*

An indirect reference is a character expression that evaluates to a filename. You must use an operator in the expression so that dBASE IV knows the character string is an expression, not the literal filename. For example, an indirect reference may be used in the CREATE command in place of a filename. If the variable Mfile contains the character string "Terms", CREATE (Mfile) and CREATE RTRIM(Mfile) will create a database file named Terms. These commands are the same as CREATE Terms. CREATE Mfile + "01" will create a database file named Terms01.

An indirect reference is similar to using the macro substitution character, as CREATE &Mfile, but operates much faster.

In the following example, the first USE command will call in the compiler at runtime, while the second USE command will not:

```
. Mfile = "Client"
. USE &Mfile
. USE (Mfile)
```

The CREATE/MODIFY QUERY/VIEW, CREATE/MODIFY LABEL, CREATE/MODIFY REPORT, CREATE VIEW FROM ENVIRONMENT, and SET CATALOG commands save the names of one or more currently open files, so that these files later can be opened automatically.

CREATE/MODIFY QUERY/VIEW creates a query (.qbe) file, which can extract records matching specified conditions, or an update query (.upd) file, which can modify the records of a database file. When you build reports or labels with CREATE/MODIFY REPORT or CREATE/MODIFY LABEL, you can save the settings of system memory variables in a print form (.prf) file. The name of the .prf file is saved in the corresponding report design (.frm) file or label design (.lbl) file. CREATE VIEW FROM ENVIRONMENT builds a view (.vue) file, which may contain the names of database files, index files, and format files. SET CATALOG creates a catalog (.cat) file, which contains the names of database and view files, and their associated index, format, label, and report files.

Sometimes the filenames saved in the .qbe, .upd, .frm, .lbl, .vue, or .cat files are written with their drive and path names, and sometimes they are not. This depends on whether the filename can be found on the default drive and in the current directory, and whether you provided an explicit drive and path as part of the filename when building the .qbe, .upd, .frm, .lbl, .vue, or .cat files.

You can provide an explicit drive and path in either of two ways:

1. You can enter the drive and path as part of the command line. For example:

```
. USE C:\DBASE\MYDATA\Myfile
```

gives the drive, C:, as well as the path, \DBASE\MYDATA, where the file can be found.

2. You can use the query clause, which is a question mark, navigate to the location of the file on disk, and then choose the file. Once the file is opened, it is as if you opened it with an explicit drive and path as part of the filename.

dBASE IV determines whether to save the filename with the full drive and path names according to the following conditions:

1. If you open a file on the default drive and in the current directory, these commands listed above store filenames without the drive and directory information, even if you provide a drive and path as part of the filename. For example, suppose the default drive is C:, and the current directory is \DBASE\MYDATA. If a catalog is open with SET CATALOG, and you USE C:\DBASE\MYDATA\Myfile, the file is saved in the catalog as Myfile.dbf. Of course, the filename will also be saved as Myfile.dbf if you typed USE Myfile.
2. To open a file that is not on the default drive and in the current directory, you may specify the drive and directory as part of the filename, or set up a search path with SET PATH.
   a. If you specify the drive and path as part of the filename, the drive and path information is saved as part of the filename. So, if the current drive is still C:, and the current directory is still \DBASE\DATA, and you type USE A:\YOURDATA\Yourfile, the file is saved in the catalog as A:\YOURDATA\Yourfile.dbf.
   b. If you SET PATH, the filename is saved just as you enter it. The following commands:

```
. SET PATH TO A:\YOURDATA
. USE Yourfile
```

   save the filename in the catalog as Yourfile.dbf.

   The following commands:

```
. SET PATH TO A:\YOURDATA
. USE A:\YOURDATA\Yourfile
```

   save the filename in the catalog as A:\YOURDATA\Yourfile.dbf.

So, the .qbe, .upd, .frm, .lbl, .vue, and .cat files never contain filenames with drive and path names, if the file can be found on the default drive and in the current directory. These .qbe, .upd, .frm, .lbl, .vue, and .cat files contain filenames with drive and path names, if the file is on another drive or in another directory *and* if you provide the explicit drive and path names.

## Aliases

dBASE IV allows up to ten database files to be open simultaneously by allotting each open file its own unique work area.

Internally, dBASE IV keeps track of the open database files by their *aliases.* An alias can be an alias name (which may be the same as its filename), a work area letter, or a work area number.

If you provide an alias name with the ALIAS option of the USE command, you may use this alias name as an abbreviation in place of the database filename when referencing the work area. If you don't assign an alias name with the ALIAS option of the USE command, dBASE IV uses the filename as the default alias name.

Work area letters are the letters from A to J or a to j. Work area A is the first work area; J is the tenth work area.

Work area numbers are from 1 to 10. In earlier dBASE versions, you could use work area numbers only in the SELECT command. In dBASE IV, you can use a work area number in any command that accepts an alias.

Some valid filenames, such as X-y, cannot be used as aliases because they contain characters that dBASE reserves for other uses. If a filename contains characters that prohibit it from being used as the default alias name, and if you do not provide another alias name with the ALIAS option of the USE command, dBASE IV assigns a letter alias, from A to J, as the default.

You may use an indirect reference to an alias, just as you may use an indirect reference to a filename. If you include an operator in the alias character string, dBASE IV knows that the string is an expression, not the literal alias. dBASE IV also evaluates strings that contain an alias symbol (->) or square brackets ([]) because these symbols indicate a reference to a field, memory variable, or array element, rather than to a literal alias. In these cases, dBASE IV evaluates the expression for a work area alias name, work area letter, or work area number. For example,

```
. Expwa = 3
. GO 5 IN (Expwa)
```

positions the record pointer to the fifth record in work area 3.

Indirect alias references are similar to using the macro substitution character, such as SKIP 2 IN &Expwa, but operate much faster.

## Programs and Procedures

If you execute the same series of commands repeatedly, and you want to automate the process, you may save these instructions in a program file.

### Programs

A program is a sequence of dBASE IV commands contained in a disk file. When you execute the program file, the commands in the program are executed as if you had typed them from the dot prompt.

You can use the dBASE IV program editor, which is accessed with MODIFY COMMAND, to write and save your programs. MODIFY COMMAND creates a disk file with a .prg extension, or a .prs extension for an SQL program.

When creating a program file, press ↵ to indicate the end of a command line. To continue a command on several lines, type a semicolon at the end of each line except the last.

You can execute the program file with the DO command. Before executing your program file, DO compiles the commands into *object code*, which runs much faster than the original *source code* in the .prg or .prs file, and it writes the object code to a disk file with a .dbo extension. You may also use the COMPILE command to generate an object file without executing the program. dBASE IV notifies you of any errors it encounters during compilation.

### Procedures

A program may be composed of one or more routines, called *procedures*. Each procedure usually does one basic task, and can be called from other procedures, programs, or from the dot prompt with a DO command. When the procedure finishes its task, it returns control to the program or procedure that called it, or to the dot prompt.

In earlier dBASE versions, procedures were usually contained in a separate file called a *procedure file*. You opened one procedure file at a time, containing up to 32 procedures, with the SET PROCEDURE command.

dBASE IV handles procedures differently. You can incorporate them directly into a program file, or put them into a separate procedure file. The program or procedure file can contain as many procedures as available RAM permits, up to a maximum of 963 procedures per file. Each procedure must begin with the keyword PROCEDURE.

dBASE IV maintains a procedure list at the beginning of every object (.dbo) file. It treats the main program itself as a procedure, giving it a default procedure name that matches the source program filename. This procedure name is the first one in the procedure list. Each subsequent procedure, whether contained in the current source file or in a separate procedure file, is added to the list when the source file is compiled to an object file.

For example, suppose you have the following hypothetical program, Main:

```
*Main.prg
<commands>
DO A
DO B
DO C
RETURN


PROCEDURE A
<commands>
RETURN


PROCEDURE B
<commands>
RETURN


PROCEDURE C
<commands>
RETURN
```

dBASE IV will include four procedures in the procedure list for the compiled object file: Main (the default procedure name), A, B, and C. Note that only the code at the beginning of a program file is assigned the default procedure name. Any loose code following RETURN and before PROCEDURE will be compiled, but will cause a warning error during compilation. As this code will not be executed by the DO command, the compiler verifies that you want this code embedded in your program and object code files.

Putting the procedures in the main program file eliminates the need for many separate procedure files. This makes programs run more quickly, since dBASE IV does not have to open and close these procedures before running them. You can still use SET PROCEDURE TO <procedure filename> for procedures not activated with the DO command.

When dBASE IV encounters a DO <procedure name> command in program code, it searches for the named procedure in the following order:

1. Search the currently executing object (.dbo) file.
2. Search the SET PROCEDURE file, if one is active.
3. Search other open object files (most recently opened first).
4. Find and open an object (.dbo) file of that name.
5. Find and compile a program (.prg) file of that name.
6. Find and compile an SQL program (.prs) file of that name.

Because dBASE IV uses this search order, you can hide procedures of the same name from each other. You can also still use the SET PROCEDURE command to open a procedure file for the DO command search.

The dBASE IV procedure limits are:

- 64K of compiled code per procedure
- 32 active object (.dbo) files, including a file opened with SET PROCEDURE
- 963 procedures per object file

Some dBASE IV commands also compile object code from source code generated by a design screen. For example, REPORT FORM will compile an .frg file, which was created by CREATE REPORT, into an .fro file containing object code. Since dBASE IV handles procedures in object file form, it does not distinguish between procedures created by DO, COMPILE, and DBLINK, nor between object code in format (.fmo), report (.fro), label (.lbo), or query (.qbo) files. It also does not distinguish between object code generated from .prg files and SQL program (.prs) files. Any object code procedure may be called and added to the procedure list.

## Using Commands

This section discusses required and optional parts of a command line, and includes rules for building expressions.

The structure of a command line is called its *syntax*. Each command line begins with a verb, and many commands also have one or more clauses that tailor the command to meet a need. The general syntax of a command is described below.

**NOTE**
*You will find many exceptions to the general syntax paradigm given below. Not all commands use all the options given in this paradigm. The exceptions are covered in the alphabetical listings. Read each entry in the subsequent chapters carefully, before using any language component.*

```
<command verb> [<expression list>] [<scope>]
    [FOR <condition>] [WHILE <condition>] [TO FILE <filename>
    /TO PRINTER/ TO ARRAY <array list>/TO <memvar>]
    [ALL [LIKE/EXCEPT <skeleton>]] [IN <alias>]
```

<command verb> is the name of the dBASE command.

[ ] (square brackets) indicate that the item is optional.

< > (angle brackets) indicate that you must supply a specific value of the type required for the item in the brackets.

/ (slash) indicates an either/or choice.

**NOTE**
*Do not type the square brackets, angle brackets, or slash when entering a command.*

< list > means a group of like items separated by commas.

[ < expression list > ] is one or more expressions, separated by commas. They do not have to be the same data type (see the "Expressions" section below).

[ < scope > ] indicates the number of records the command can access. The keywords for scope are:

- RECORD < n > to specify a single record by its number
- NEXT < n > for n records beginning with the current record
- ALL for all the records in the database
- REST for records from the current one to the end of the file

If a command accepts a FOR or WHILE clause, however, the conditions you specify in these clauses have precedence over < scope >.

< condition > is a comparison between two or more items like *Name = "Smith"* or a logical statement like .NOT. EOF().

[FOR < condition > ] tells dBASE IV that the command applies only to records that meet the condition. If you use FOR, dBASE IV rolls the record pointer back to the top of the file, and compares each record with the FOR condition.

[WHILE < condition > ] begins processing with the current record in the database file, and continues for each subsequent record as long as the condition is true.

[TO ...] controls the output of the command. Certain commands allow you to send the output to a file, a printer, a designated array, or a memory variable.

*Memvars* (or *memory variables*, or just *variables*) are data values you temporarily store in RAM. You assign each of these values a name so that you can later retrieve it from memory by name. Use these values to perform calculations, comparisons, and other operations. You create memory variables with any of the following commands: ACCEPT, AVERAGE, CALCULATE, COUNT, INPUT, PARAMETERS, PRIVATE, PUBLIC, STORE, SUM, and WAIT.

The DECLARE command creates a special set of memory variables called an *array*. An array is a one- or two-dimensional table of values stored in memory. Each entry in the array is called an *element*, and each element in an array may be treated like a memory variable, and may be used in an expression.

Commands that allow output to a memvar also allow output to an array element.

[ALL [LIKE/EXCEPT < skeleton > ]] directs dBASE IV to include or exclude the files, fields, or memory variables that match the *skeleton*. The skeleton is a general pattern that filenames, fields, or memory variable names may match. You may use the ? and * symbols as wildcards in the skeleton. A ? represents any single character, while * represents a group of any characters.

[IN ...] allows you to manipulate the database file in another work area without SELECTing it as the current work area. The IN clause may contain the alias name, letter, number, or an expression that evaluates to an alias name, letter, or number. The USE command, however, requires a work area number in the IN clause because no other alias exists until after the file is opened.

< filename > may be the actual name of a file as it is written on disk, or an indirect reference to the filename. Filenames are discussed earlier in this chapter.

## Expressions

An expression is formed with combinations of:

Field names
Memory Variables
Array Elements
Constants
Functions
Operators
System Memory Variables

So far in this chapter, you have already encountered memory variables, array elements, and functions.

A *field name* is the name of one *field*, or item of information, contained in every *record* of a database file. Lastname might be the field name of a field that contains clients' last names. Each record in the database file would typically have one client's last name entered in the Lastname field.

If the field is not in the currently-selected database file, you must qualify the field name with the alias name. Use the alias symbol (->) between the field name and alias name. Note that you enter the alias symbol with two keystrokes, a hyphen (-) and a greater-than symbol (>). For example,

```
Client->Lastname
```

means the Lastname field in the database file whose alias is Client.

**NOTE**
*When fields and memory variables have the same name, fields take precedence over memory variables. You can change this by preceding the memory variable name with the memory variable alias symbol,* M->.

A *constant* is a literal value embedded right in the expression, such as "$" (a character constant), or 2 (a numeric constant).

*Operators* are symbols that link memory variables, fields, constants, and functions so that the dBASE IV processor can evaluate the entire expression as one unit. The types of operators are discussed later in this chapter.

When you combine fields, memory variables, constants, or a function's returned value in one expression, they must be the same *data type*. See the discussion of data types in the next section. If necessary, use functions to convert elements of differing data types to one common type. For example, you must use functions to convert numeric variables to character data type before joining them with character constants. The expressions in an expression list, however, do not have to be of the same data type.

## Data Types

There are four data types: *character*, *numeric*, *date*, and *logical*. There are really two numeric types, but no conversion is needed between these two. The data types are discussed below.

The abbreviations for dBASE IV data types are:

- <expC> for character type
- <expN> for Binary Coded Decimal (BCD) numeric type
- <expF> for floating point binary numeric type
- <expD> for date type
- <condition> for logical type

### Character Type

Character type fields, constants, and variables contain character strings. Character constants must be bounded with delimiters, such as double quotes (" "), single quotes (' '), or square brackets ([ ]). You may also store a decimal sequence to a character string with the CHR() function. For example:

```
. STORE "A" TO Mletter
```

and

```
. STORE CHR(65) TO Mletter
```

both create a character type memory variable containing the letter *A*.

## Numeric Types

dBASE IV supports two numeric data types: *type N* and *type F*. Type N numbers are Binary Coded Decimal (BCD) numbers. Type F numbers are the floating point binary numbers that were used in dBASE III PLUS. The distinction between these two types is internal to dBASE IV; both forms look the same when you display them.

As type N numbers contain a decimal representation, they are not subject to rounding errors. They are useful in business and financial applications where totals must balance. Type F numbers are more useful in scientific applications, when you are dealing with very large or very small numbers, or when performing repeated multiplication or division. Type F numbers, however, may yield an approximate result when rounded or truncated. Therefore, they are not as useful as type N numbers in business and financial applications.

Numbers that you input into dBASE IV are type N by default. You may use the FLOAT() function to convert these numbers to type F. If you define a database field as type F, however, you may enter type F numbers directly into this field without using the FLOAT() function for conversion.

Numeric fields imported from dBASE III PLUS are converted to type N numbers.

If a function or command returns a number, its number type depends on the function or command and the input. The functions EXP(), LOG(), SQRT(), and all the trigonometric functions always return a type F number. All other functions return either a type N number or the same type as the input. Note that operations combining different number types output type F numbers.

If you have very large or very small numbers, both type F and type N numbers display in scientific notation. The exponent is preceded by the letter *E*, such as E+09.

## Date Type

Use date type fields and memory variables to store calendar dates. The size of a date variable or field is always eight bytes, and the total memory requirement is nine bytes. dBASE IV validates date variables whenever they are entered or changed. The default date format is the American style, mm/dd/yy. You can change the format with SET DATE, or with the DATE setting in Config.db. See Chapter 6, "Customizing dBASE IV," to change Config.db parameters.

dBASE IV provides a set of delimiters that identify date values. These are {}, referred to as curly braces, and are equivalent to the CTOD() function. For example, {12/20/59} is the same as CTOD("12/20/59").

A date may be subtracted from another date. The result is a number (the number of days between the dates). A number (representing a number of days) may be added to or subtracted from a date. The result will also be a date.

### Logical Type

Logical fields and variables are stored as true (.T.) or false (.F.). Logical fields or variables will accept T, t, Y, or y for true and F, f, N, or n for false. When you create them from the keyboard, you must delimit logical values by periods (for example, STORE .T. TO Mlogic). A logical expression is also called a *condition*.

## Operators

dBASE IV provides four types of operators: *mathematical*, *relational*, *logical*, and *string*.

### Mathematical Operators

Mathematical operators generate numeric results.

| | |
|---|---|
| + | Addition/Unary Positive |
| − | Subtraction/Unary Negative |
| * | Multiplication |
| / | Division |
| ** or ^ | Exponentiation |
| () | Parentheses for grouping |

### Relational Operators

Relational operators generate logical results, that is, true (.T.) or false (.F.). You can use relational operators with character, numeric, date, or logical expressions. Both expressions you use in a relational operation must be of the same type.

| | |
|---|---|
| < | Less than |
| > | Greater than |
| = | Equal to |
| < > or # | Not equal to |
| < = | Less than or equal to |
| > = | Greater than or equal to |
| $ | Substring comparison (For example, if A and B are character strings, A$B returns a logical true if A is either identical to B or contained within B.) |

### Logical Operators

Logical operators obtain a logical result from comparing two expressions.

| | |
|---|---|
| .AND. | Logical and |
| .OR. | Logical or |
| .NOT. | Logical not |
| () | Parentheses for grouping |

### String Operators

String operators concatenate two or more character strings into a single character string.

| | |
|---|---|
| + | Trailing spaces between the strings are left intact when the strings are joined |
| − | Trailing spaces between the strings are moved to the end of the last string |
| () | Parentheses for grouping |

## Precedence of Operators

Each type of operator has a set of rules that governs the order in which operations are performed. These are called the *order of precedence* for the operator.

Relational and string operators have only one level of precedence, and are performed in order from left to right.

### Mathematical Operators

The precedence levels for mathematical operators are:

1. Unary + (positive) and unary − (negative) signs
2. Exponentiation
3. Multiplication and division
4. Addition and subtraction

### Logical Operators

The precedence levels for logical operators are:

1. .NOT.
2. .AND.
3. .OR.

### Combinations of Operators

When several of the four types of operators are used in the same expression, the precedence levels are:

1. Mathematical or string
2. Relational
3. Logical

All operations at the same precedence level are performed in order from left to right. Use parentheses to override the order in which operations are performed. Operations within the inner nested parentheses are performed first.

## Using SET Commands

SET commands control dBASE IV's system parameters from the dot prompt or from within program and procedure files. Chapter 3 lists and annotates all the dBASE IV SET commands. Some of these commands have default settings installed when you start dBASE IV. These defaults are listed in Table 6-4 in Chapter 6, "Customizing dBASE IV." You can also learn how to change these default settings in Chapter 6.

You can change any of the SET commands from the dot prompt. These are only in use temporarily, however, as all SET parameters revert to their defaults when you QUIT.

There are two common syntaxes for SET commands:

```
SET <parameter> ON/OFF
```

or

```
SET <parameter> TO <expression>
```

## Using Functions

Functions perform specialized operations that augment and enhance the dBASE IV commands. Functions evaluate or convert data, and then return a result.

dBASE IV functions all have parentheses after the function name, except for the macro substitution function, which is described in the next section. The parentheses may or may not contain parameters to be evaluated.

Chapter 4 lists and annotates the dBASE IV functions.

### User-Defined Functions

dBASE IV allows you to define your own functions. With a user-defined function (UDF), you can further customize your database operations to fit your needs.

A UDF is a procedure. It begins with the FUNCTION command, and contains commands and a parameter list that it uses to return a value. The UDF filename cannot be an existing dBASE IV function or command.

The syntax for a user-defined function is:

```
<UDF filename>([<parameter list>])
```

The <UDF filename> you select is the user-defined function procedure file. The optional <parameter list> passes parameters to the UDF procedure file. You must use the RETURN command in the procedure file to return a value generated by your UDF.

## Commands Not Allowed in User-Defined Functions

You can reference most commands in UDFs without restrictions; others, however, cannot be used at all.

Two commands you can use with restrictions are CLEAR and READ: CLEAR if it has no arguments, READ if there is no active format file.

The following commands are not allowed in UDFs. If you use any of them, an error message appears.

- APPEND
- APPEND FROM
- APPEND FROM ARRAY
- APPEND MEMO
- ASSIST
- BEGIN TRANSACTION/ END TRANSACTION
- BROWSE
- CANCEL
- CHANGE
- CLEAR ALL/FIELDS
- CLOSE ALTERNATE
- CLOSE FORMAT
- CLOSE PROCEDURE
- COMPILE
- CONVERT
- COPY
- COPY FILE
- COPY INDEXES
- COPY MEMO
- COPY STRUCTURE
- COPY STRUCTURE EXTENDED
- COPY TAG
- COPY TO ARRAY
- CREATE or MODIFY STRUCTURE
- CREATE FROM
- CREATE VIEW FROM ENVIRONMENT
- CREATE/MODIFY APPLICATION
- CREATE/MODIFY LABEL
- CREATE/MODIFY QUERY/VIEW
- CREATE/MODIFY REPORT
- CREATE/MODIFY SCREEN
- DEBUG
- DEFINE BAR
- DEFINE BOX
- DEFINE MENU
- DEFINE PAD
- DEFINE POPUP
- DEFINE WINDOW
- DELETE TAG
- DIR
- EDIT
- ERASE or DELETE FILE
- EXPORT
- HELP
- IMPORT
- INDEX
- INSERT
- JOIN
- LABEL FORM
- LOAD
- LOGOUT
- MODIFY COMMAND/FILE
- MOVE WINDOW
- ON ERROR/ESCAPE/KEY
- ON PAD
- ON PAGE
- ON READERROR
- ON SELECTION PAD
- ON SELECTION POPUP
- PACK
- PRINTJOB/ENDPRINTJOB
- PROTECT
- QUIT
- REINDEX
- REPORT FORM
- RESTORE
- RESTORE MACROS
- RESTORE WINDOW
- RESUME
- ROLLBACK
- SAVE
- SAVE MACROS
- SAVE WINDOW
- SET
- SORT
- SUSPEND
- TOTAL
- TYPE
- UPDATE
- ZAP

In addition to the full-screen SET command, the following SET commands cannot be included in a UDF:

| | | | |
|---|---|---|---|
| CATALOG | FORMAT | SKIP | TRAP |
| DEBUG | PROCEDURE | SQL | VIEW |
| DEVICE | RELATION | STEP | WINDOW |
| FIELDS | | | |

You may not include macro substitution in a user-defined function, but you may include indirect file references and indirect alias references.

## Macro Substitution

The macro substitution (&) function instructs dBASE IV to retrieve the *contents* of a character memory variable, and not the memory variable name itself. To use it, simply place the & symbol before the memory variable name:

```
. STORE "Lastname, Firstname" TO Mfields
. LIST &Mfields
```

In this example, LIST &Mfields is equivalent to LIST Lastname, Firstname.

The macro substitution function allows you to prompt the user for a piece of information in a program, and immediately use that information as part of a command. With macro substitution, you can build part of a command and allow the user to supply certain arguments.

When you use a macro, dBASE IV assumes that the command line cannot be compiled at compile time. When the command is encountered at runtime, it is expanded using the current value of the memory variable following the &, and the line must be recompiled. This slows down processing time.

dBASE IV allows you to use a macro with a variable in place of a command, such as:

```
    . Command = [RESET IN 1]
    . &Command
```

If you do this, of course, dBASE IV must compile the command every time it executes it. Depending on the command, this may require calling in the full compiler once again.

RunTime applications, however, cannot use macros in place of commands.

If you use macro expansion only in expressions, most of the command can be compiled at compile time, and there is no need to call in the compiler again at runtime.

You can speed up processing time by using an indirect reference to a filename rather than macro substitution. (Indirect references are discussed earlier in this chapter, in the "Filenames" section.)

For more examples of how macro substitution works, see the discussion of the & function in Chapter 4, "Functions." Don't confuse macro substitution with macros, in which you store a series of keystrokes.

## Using System Memory Variables

System memory variables are memory variables that dBASE IV automatically creates and maintains. They control the appearance of printed and screen output and also store printer settings. More specifically, they control:

- Characteristics of print jobs, such as form feeds and the number of copies printed
- The appearance of paragraphs, such as alignment, indentation, and margins
- The appearance of the printed page, such as the print pitch, print quality, page length, and page left offset

The names of all system variables start with an underscore (_) character, to distinguish them from ordinary memory variables. (You may not define any other memory variables starting with an underscore.)

Upon start-up, dBASE IV automatically initializes system variables to their defaults. You can change their values through the reports design and labels design screens, at the dot prompt, or in a program. The CLEAR MEMORY and RELEASE commands do not remove system memory variables from memory.

System memory variables follow the normal scoping rules for memory variables. When you have finished running a program containing privately declared system variables, the variables automatically revert to their original settings. You cannot RELEASE these variables.

Many system memory variables work on the data stream that is being output. This *streaming output* is produced by all commands that create output, except the @, @...TO and EJECT commands. Streaming output destinations may be controlled by the SET CONSOLE, SET PRINTER, and SET ALTERNATE commands, and by the TO PRINTER/TO FILE <filename> options of commands such as LIST/DISPLAY.

Streaming output starts at the current cursor position on the screen, or at the current printhead position on the printer, or at the current file pointer position if the data is being sent to a disk file. Even though @ and @...TO commands do not produce streaming output, they may affect the positioning of subsequent output. For example, changing the current cursor position on the screen with an @ command affects where a later ?/?? command will display.

The system memory variables that act on streaming output are _box, _pageno, _pcolno, _pform, _plength, _plineno, _pspacing, and _tabs.

Other system memory variables that do not act on streaming output may be classified as *printer-specific*, *printjob-specific*, or *paragraph-specific*.

Printer-specific system memory variables control printer settings. These are _padvance, _pdriver, _ploffset, _ppitch, _pquality, and _pwait.

Printjob-specific system memory variables are activated by the PRINTJOB command. These are _pbpage, _pcopies, _pecode, _peject, _pepage and _pscode.

Paragraph-specific system memory variables affect the formatting of text. These are _alignment, _indent, _lmargin, _rmargin, and _wrap.

You can see the relationships among several system memory variables in Figure 1-1. These system memory variables are:

- _plength; page length
- _ploffset; page offset from left edge
- _lmargin; left margin from _ploffset
- _rmargin; right margin column number
- _indent; paragraph indent

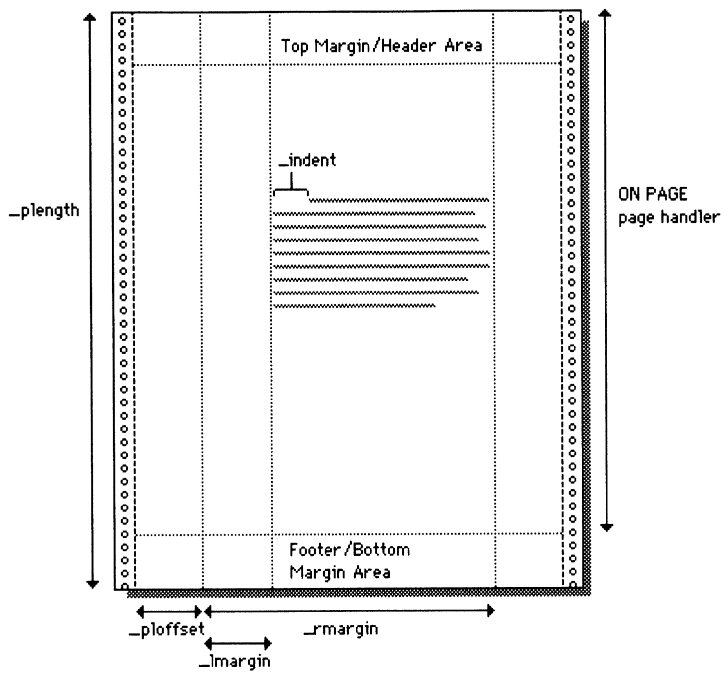


Figure 1-1 Page layout for typical printjob

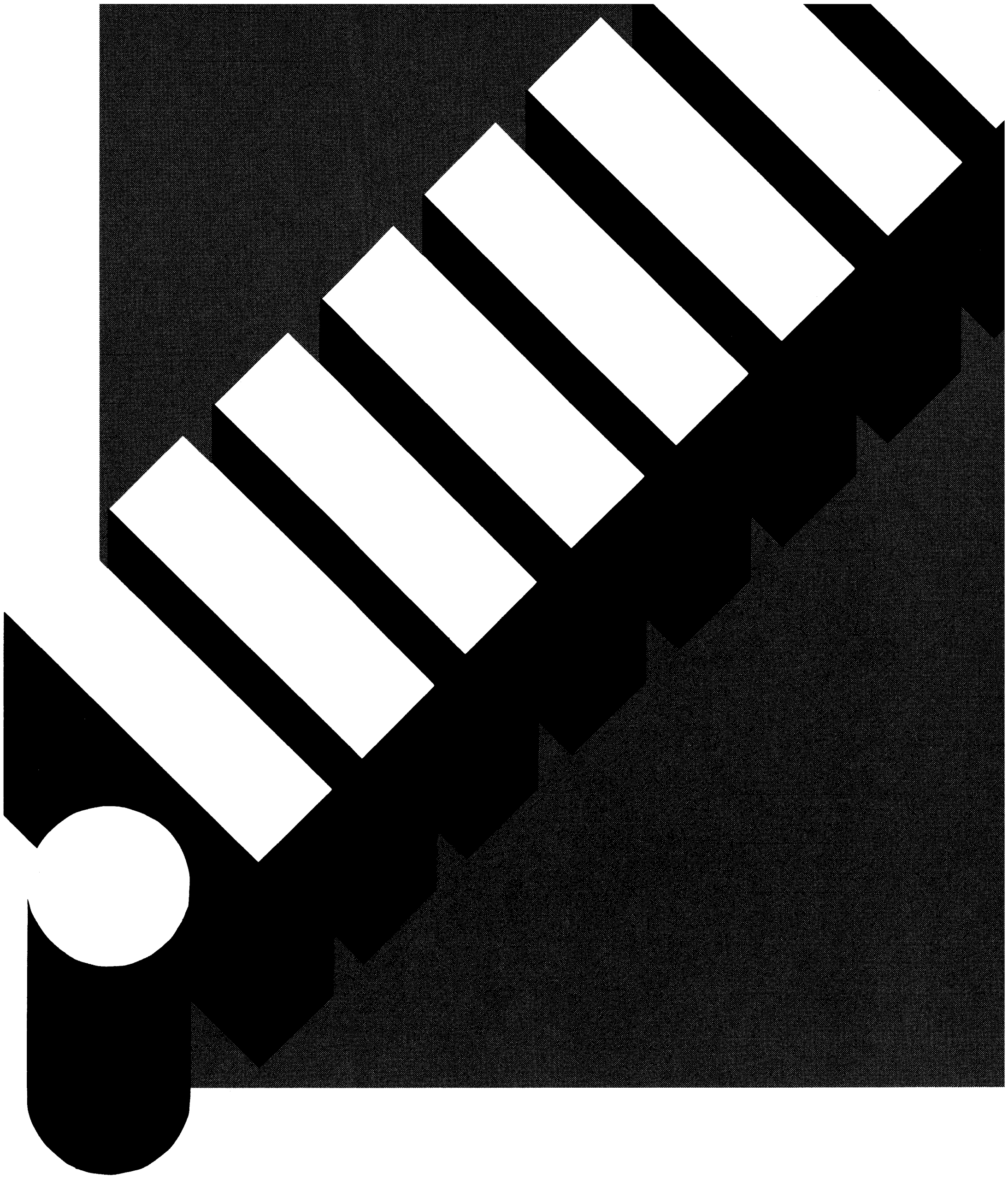
The ON PAGE footer and header procedures also affect the printed page. See the ON PAGE description and page handler examples in Chapter 2, "Commands."

## Print Form File

The reports design and labels design screens handle system variables for you. When you create a report (CREATE/MODIFY REPORT) or label (CREATE/MODIFY LABEL), you can store the changed system variable definitions to a binary *print form file*. This file normally has the same name as the report or label, with the file extension .prf.

dBASE IV activates this print form file when you next modify the report or label, or when you select **Use print form** from the **Print** Menu. The REPORT FORM command, however, does not automatically activate the print form. You must determine print settings prior to issuing REPORT FORM by setting _pform equal to the print form filename, or by making changes to individual system variables.

# Commands



i 1 **2** 3 4 5 6 A B C

D E F G In

# Commands

## ?/??

?/?? displays the value of one or more expressions. ?/?? roughly translates to: "What is . . . ?" or "What is the value of . . . ?" or, simply, "Print . . ."

### Syntax

```
?/?? [<expression 1> [PICTURE <expC>]
    [FUNCTION <function list>] [AT <expN>]
        [STYLE <font number>]]
    [,<expression 2> ...] [,]
```

### Usage

If SET PRINT is ON, the output of the ? command is sent to the printer.

The single question mark command issues a carriage return and line feed before displaying the results of the expression list. The double question mark does not.

?? displays the expression list starting at the current cursor or printer position.

You can enter as many expressions and clauses as will fit in the 1024-character command line.

### Options

The PICTURE or FUNCTION, AT, and STYLE options let you customize the appearance of printed reports.

PICTURE templates and functions format the output. All the templates work with the ?/?? command. See the @ command for a description of templates.

Six functions that work with the ?/?? command handle *long fields*, which are fields whose contents exceed the PICTURE template, and *short fields*, which are fields whose contents do not completely fill up the PICTURE template.

Two of these functions, H and V, are available only for use with the ?/?? command. Precede these functions with the FUNCTION or PICTURE keyword, as shown under "Special Case," below.

@ H lets the field stretch horizontally to accommodate data.
@ V <n> lets the field stretch vertically to accommodate data.

<n> is the maximum number of columns or rows in the field display. Using the V or H function, you can modify the height or width of the display. The H function stretches long fields horizontally to the right, pushing ahead any text on the same line. It shrinks short fields to the left, also pulling other text on the same line to the left. It will not pull other text on the line, however, if a column is specified with the AT option. For word wrapping to work with the H function, the _wrap system memory variable must be set to true (.T.). The V function stretches long fields vertically down the page, in columnar fashion.

You can limit how far a field stretches by indicating the maximum number of columns allowed for the field. Any additional characters will be truncated. Without the <n>, the field will grow to accommodate its entire contents.

If the command line ends with a comma, the first line of any vertically-stretching fields will output, but subsequent lines of the vertical field are kept internally and are not output until either a RETURN, ENDPRINTJOB, or ?/?? (single or double question mark command line not ending with a comma) is encountered. This allows you to output many fields across a line (as for a wide-carriage printer) with several ?? commands, and to control when to output the second and subsequent lines of each vertically-stretching field.

If you don't specify the H and V functions, the field width is fixed by the template, and extra characters are truncated. The field height defaults to one line.

The other four functions align short fields, and can also be used with the @ command.

| | |
|---|---|
| @ B | Left-aligns text within a field. |
| @ I | Centers text within a field. |
| @ J | Right-aligns text within a field. |
| @ T | Trims leading and trailing blanks from a field. |

Short fields contain text that does not completely fill the defined PICTURE template. If you want to trim a field before aligning it, list the T function along with one of the other three functions. If you specify no alignment, strings are left-aligned and numbers right-aligned.

The ?/?? command also supports two other functions, $ and L. The $ function displays a floating currency symbol before or after the amount. If SET CURRENCY is LEFT, the symbol displays just before the amount. If SET CURRENCY is RIGHT, it displays just after the amount. The L function displays a short field with leading zeros.

AT specifies the column at which the expression displays. Use this option to print columns of text which must line up, regardless of the length of printed text to the left of the column.

STYLE prints the text in various styles, such as bold or italic. Depending upon your monitor, STYLE may not change the output displayed on the screen, but will affect the printed output. The STYLE clause can consist of letters, numbers, or a combination of the two.

The allowed letters are:

| | | |
|---|---|---|
| B | — | bold |
| I | — | italic |
| U | — | underline |
| R | — | raised (superscript) |
| L | — | lowered (subscript) |

The allowed numbers are 1 through 5.

The numbers correspond to fonts that you have previously defined in Config.db, using the PRINTER setting (see Chapter 6, "Customizing dBASE IV").

You may combine different styles, and print different text on the same line in different styles (see "Examples," below).

You can also use the ??? command, and the system memory variables _pscode and _pecode, to change typestyles. In general, use ?/?? with its STYLE option to change typestyles of individual text items, ??? to change typestyles on a broader basis within a document, and _pscode and _pecode to define the overall typestyle for a document. (See Chapter 5 for information on the system memory variables.)

## Tips

? without an expression displays a blank line, and can be used to skip a line in the output. To single- or double-space the output, however, use the _pspacing system memory variable.

## Special Case

The alignment and stretch functions discussed above interact with the _wrap system variable in the case of printing memo fields. If _wrap is true and @ H is used, rewrapping of memo fields on printout occurs within the margins specified in the system memory variables _lmargin and _rmargin. (This section refers to a number of the dBASE IV system variables. See Chapter 5, "System Memory Variables," for details.)

If you stretch the field horizontally with the @ H function, dBASE IV ignores the margins embedded in the memo field, and instead wraps the text between the _lmargin and _rmargin.

If you vertically stretch a field with @ V, the text inside it can wrap. But the field is still affected by the PICTURE template, if any, and text may be truncated at the end if the text string is longer than the template allows.

In either case, dBASE IV honors paragraph breaks, paragraph indentation, and alignment contained in the memo field.

Use the B, I, or J function to override an overall _alignment setting, or individual paragraph alignments, in a memo field. The following routine vertically centers a memo field, named *Notes*, and stretches it within the 15-character display column defined by a PICTURE template. The vertical field centering overrides the initial _alignment = "LEFT" setting.

```
_alignment = "LEFT"
_wrap = .T.
? Notes PICTURE "!XXXXXXXXXXXXXX" FUNCTION IV15
```

If you had stretched the field horizontally rather than vertically in the previous example, the text would be centered between _lmargin and _rmargin.

If _wrap is set to false (.F.) when you print a memo field using the H function, the memo field starts printing at the current column (PCOL() or _pcolno), and that column becomes the effective left margin. dBASE IV honors any page breaks, margins, and indentation contained in the paragraphs.

## Examples

From the Client database file, to print the Client_id and the Client field values in bold and the value of Lastname in italics:

```
. USE Client
. SET PRINT ON
. ? Client_id STYLE "B", Client STYLE "B", Lastname STYLE "I"
A00001 WRIGHT & SONS, LTD         Wright
```

Overstriking text is sometimes useful to show changes made to a document. To overstrike a line of text from a program file, use the AT option with _wrap set to false (.F.). (Chapter 5 discusses _wrap and the other system memory variables.)

```
_wrap = .F.
SET PRINT ON
? "Pending receipt of file."
?? "////////////////////////File received 3/7/88" AT 0
```

To *overwrite* rather than *overstrike* text, use the technique just shown with _wrap set to true.

## See Also

???, @, SET PRINT

Chapter 5, "System Memory Variables"

Chapter 6, "Customizing dBASE IV"

# ???

??? sends output directly to the printer, bypassing the installed printer driver.

## Syntax

??? <expC>

## Usage

You may use the ??? command to send characters to your printer that will not change the printer's current row and column position. You usually send printer control codes when the printer driver does not support a particular printing capability.

The ?/?? command and the system memory variables _pscode and _pecode also send printer control codes to the printer. In general, use ?/?? (STYLE option) to change typestyles of individual text items, ??? to change typestyles on a broader basis within a document, and _pscode and _pecode to define the overall typestyle for a document.

Printer control codes are specific to the printer you are using. Consult your printer manual for the necessary control codes.

Printer control codes may include any printable character except the double quote mark ("), as well as non-printable characters, such as **Esc**. You can define these non-printable characters in a variety of ways.

Control character specifiers are strings that identify non-printable characters. You must include the curly braces ({}).

Table 2-1 Control character specifiers

| ASCII Code | Control Character Specifier |
|---|---|
| 0 | {NULL} or {CTRL-@} |
| 1 | {CTRL-A} |
| 2 | {CTRL-B} |
| 3 | {CTRL-C} |
| 4 | {CTRL-D} |
| 5 | {CTRL-E} |
| 6 | {CTRL-F} |
| 7 | {BELL} or {CTRL-G} |
| 8 | {BACKSPACE} or {CTRL-H} |
| 9 | {TAB} or {CTRL-I} |
| 10 | {LINEFEED} or {CTRL-J} |
| 11 | {CTRL-K} |
| 12 | {CTRL-L} |
| 13 | {RETURN} or {CTRL-M} |
| 14 | {CTRL-N} |
| 15 | {CTRL-O} |
| 16 | {CTRL-P} |
| 17 | {CTRL-Q} |
| 18 | {CTRL-R} |
| 19 | {CTRL-S} |
| 20 | {CTRL-T} |
| 21 | {CTRL-U} |
| 22 | {CTRL-V} |
| 23 | {CTRL-W} |
| 24 | {CTRL-X} |
| 25 | {CTRL-Y} |
| 26 | {CTRL-Z} |

(*continued*)

Table 2-1 Control character specifiers (*continued*)

| ASCII Code | Control Character Specifier |
|---|---|
| 27 | {ESC} or {ESCAPE} or {CTRL-[} |
| 28 | {CTRL-\} |
| 29 | {CTRL-]} |
| 30 | {CTRL-^} |
| 31 | {CTRL-_} |
| 127 | {DEL} or {DELETE} |

## Examples

Suppose you want to send an **Esc-E** to a Hewlett-Packard LaserJet printer. (This code resets the printer.) Knowing that the ASCII code for **Esc** is 27 and an E is code 69, you can:

Use the CHR() function, which converts a number to its ASCII character equivalent:

```
. ??? CHR(27) + "E"
```

Use *control character specifiers*:

```
. ??? "{ESC}E"
```

Use entirely ASCII codes, enclosing the codes within curly braces:

```
. ??? "{27}{69}"
```

Use a combination of ASCII codes and letters:

```
. ??? "{27}E"
```

## See Also

?/??, @, CHR()

Chapter 5, "System Memory Variables"

@ is used to create custom forms for data input and output. It displays or accepts information in a specified format at a given set of screen coordinates.

## Syntax

```
@ <row>, <col>
    [SAY <expression>
        [PICTURE <expC>] [FUNCTION <function list>]]
    [GET <variable>
        [[OPEN] WINDOW <window name>]
        [PICTURE <expC>]
        [FUNCTION <function list>]
        [RANGE [<low>] [,<high>]]
        [VALID <condition> [ERROR <expC>]]
        [WHEN <condition>]
        [DEFAULT <expression>]
        [MESSAGE <expC>]]
    [COLOR [<standard>] [,<enhanced>]]
```

## Usage

<row> and <col> are numeric expressions. <row> can range from 0 to the height of the screen display, in lines. <col> can range from 0 to 79 columns. If you SET DEVICE TO PRINTER, the <row> can be from 0 to 32,767 and <col> from 0 to 255. <row> and <col> coordinates are always relative to the upper left corner of the *active* window, whether it is the global CRT or a window you've defined.

With SET STATUS ON, line 22 on the screen is reserved for the status line. If you SET STATUS OFF and do not SET SCOREBOARD OFF, line 0 is used for status information display. To free these lines for other use, you must SET STATUS OFF and SET SCOREBOARD OFF.

If you change a field's contents, and that affects the results of other calculations or field defaults defined on the form, then the calculations are updated and the results are redisplayed when you press positioning keys such as **PgUp** and **PgDn**.

The SAY keyword displays information that you do not want to change. The value of any valid dBASE IV expression can be displayed.

The GET keyword displays and allows editing of data values contained in fields, or currently assigned to memory variables or arrays. The READ command activates the GETs and a full-screen editing mode that lets you change the GET fields.

Besides using GETs with the READ command, you can create a format (.fmt) file that contains @ commands. (Format files are text files that can be created with the MODIFY COMMAND program editor or another text editor.) CREATE/MODIFY SCREEN helps you lay out an input or output screen, and generates an .fmt file that contains @ commands. You can use a format file with the APPEND, CHANGE, EDIT, INSERT, and READ commands.

You can use SET DEVICE TO PRINT to route @ commands to the printer. GETs are not routed to the printer. Most printers have limitations on the row and column coordinates. When sending @ commands to a printer, decreasing the row number in consecutive @ commands causes a page eject. Similarly, if two @ commands have the same row coordinate, the second one should have a larger column coordinate.

A user-defined function can change a value in the GET variable.

## Options

The options of the @ command are described in alphabetical order.

@ <row>, <col> — Without any options, the @ command clears the specified row beginning at the specified column position.

COLOR — Specifies the colors used for the SAY and GET variables. The <standard> color is used for SAYs, the <enhanced> color for GETs, and they follow the same rules as these options in the SET COLOR command. You can specify a foreground and background color for each one. See the SET COLOR command for information on using these codes to change the colors of the screen display.

Either the standard or enhanced colors can be left unchanged by omitting a color code. If you leave out the standard code, but want to change the enhanced colors, you should precede the enhanced code with a comma so that the command parser can determine that the standard code has not been changed. For example, to change the enhanced color to white characters on a red background:

```
. @ 2,20 GET Text COLOR ,W/R
```

The colors you specify override the SET COLOR command, but only for the output of the current @ command.

DEFAULT — Use the expression to put a preset value into a GET variable; it must match the GET variable's data type. The expression is evaluated only when you add records to a database file. The DEFAULT expression appears in the GET variable, and pressing ↵ assigns the value to the GET variable. A DEFAULT value will be overridden by any value brought forward by a SET CARRY command.

ERROR — <expC> is any valid character expression. Use this option to display your own message when the VALID <condition> is not met. Your message overrides dBASE IV's message, **Editing condition not satisfied**.

FUNCTION — The FUNCTION option is similar to the PICTURE option. See the description of PICTURE below.

MESSAGE — <expC> must be a valid character expression, which then appears when a READ is executed and the cursor is placed in the GET field associated with the message. If SET STATUS is ON, the message is centered on the bottom line of the screen. If SET STATUS is OFF, the message will not appear. This command temporarily overrides a SET MESSAGE expression.

WINDOW <window name> — When you use the @ command with GET <variable>, and the variable is a memo field, you can use the WINDOW option to open a separate editing window. Put the cursor on the memo field and press **Ctrl-Home** to open the window; **Ctrl-End** closes the window. The word "memo" you see on screen is in upper case if the field contains text. If the field is empty, the word "memo" is in lower-case letters.

The <window name> is the name of a window you have already defined.

Without the WINDOW option, dBASE IV uses full-screen editing. The editor within the window is the one you specified in the Config.db file.

OPEN WINDOW <window name> — The editing window for your memo field can be open by default. You don't need to open and close it if OPEN is included in the WINDOW option. You do use **Ctrl-Home** and **Ctrl-End** to enter and exit the window.

PICTURE — Use this option to restrict the type of data that may be entered into a variable, or to format the data displayed. The clause may consist of a *function*, which is preceded by an @ symbol and affects all the input or output characters, and/or a *template*, which affects input or output on a character by character basis. A PICTURE clause can be any character expression, although this is usually a string of characters delimited by quotation marks. If the clause is a memory variable, it is enclosed in parentheses.

The PICTURE option can also accept character expressions that are variables containing function and template symbols. Formatting contained in variables works no differently than function and template symbols used in a program or on a command line.

PICTURE functions and templates are described in greater depth in the sections below. Functions may also be used with the FUNCTION option, and these do not need to be preceded with the @ symbol. Please note that the PICTURE and FUNCTION options can each be used only once on any one command line.

RANGE — Use this option with character, numeric and date variables to specify lower and upper bounds. The <low> and <high> expressions must be the same data type. They define the minimum and maximum values that may be entered in response to the GET. The RANGE values are inclusive.

Enter only one expression if you want to specify only a lower or upper limit. You must include the comma (,) as in RANGE ,30 or RANGE 10, . The comma helps the command parser determine whether you supplied the <low> or <high> value. If you then enter an invalid number or date, dBASE IV prompts for a new value until a valid number or date is entered.

The RANGE prompt reflects whether you specified both upper and lower limits, or just one. With <m> as the upper limit you've specified, and <n> as the lower limit, the prompts are:

- **RANGE is <m> to <n>** — the data is outside the upper and lower limits.
- **Lower bound is <n>** — the data is below the lower limit.
- **Upper bound is <m>** — the data is above the upper limit.

If you specify a RANGE and press ↵ in response to a GET, no range checking is done. Therefore, if a field or variable is set to a value outside the specified range, pressing ↵ leaves the value unchanged.

An ON READERROR command, and the commands it may execute, preempts the range checking messages.

VALID — This option can state a condition that must be met before data is accepted into the GET variable. If the condition is not met, the message **Editing condition not satisfied**, or the message you defined with the ERROR option appears. No checking takes place if you press ↵ without pressing any other key.

**NOTE**
*You can enter a user-defined function as the VALID condition, if the function returns a logical value. This enables you to have quite powerful and extensive data validation. Refer to Chapter 1, "Essentials," for more information on user-defined functions.*

WHEN — You can provide a condition that is evaluated when you try to move the cursor into a GET field. If the condition is true (.T.), the cursor moves into the field for you to edit. If the condition is false (.F.), the cursor skips the field and moves to the next one.

## Format Functions

You may use most format functions with the PICTURE or FUNCTION options, or with the TRANSFORM function.

If you use a format function in a PICTURE clause, the @ symbol must appear as the first character in the clause. If you use a format function with the FUNCTION option, the @ symbol is not needed. If you use both a format function and a template in a PICTURE clause, a space must separate the two.

dBASE IV provides the following format functions:

Table 2-2 Format functions used in dBASE IV

| Function | Description |
|---|---|
| ! | Allows any character and converts letters to upper case. |
| ^ | Displays numbers in scientific notation. |
| $ | Displays data in currency format. |
| ( | Encloses negative numbers in parentheses. |
| A | Alphabetic characters only. |
| B | Left-aligns text within a field. |
| C | Displays CR (credit) after a positive number. |
| D | The current SET DATE format for dates. |
| E | European date format. |
| I | Centers text within a field. |
| J | Right-aligns text within a field. |
| L | Displays leading zeros. |
| M | Allows a list of choices for a GET variable. |
| R | Displays literal characters in the template, but doesn't enter them in the field. |
| S<n> | Limits field width display to <n> characters and horizontally scrolls the characters within it in <n> columns. <n> must be a literal positive integer. |
| T | Trims leading and trailing blanks from a field. |
| X | Displays DB (debit) after a negative number. |
| Z | Displays zero numeric value as a blank string. |

Some functions are restricted according to the data types to which they apply:

- The functions ^ , $, (, C, L, X, and Z can be used only with numeric data (either type N or type F). Furthermore, (, C, and X can be used only to display data (that is, with the SAY clause). The functions $ and L can be used with SAY or GET.
- D and E apply only to date data.
- A, M, R, and S <n> are relevant only to character data.

You can define new format functions by combining the functions in the table. For example, the function XC displays DB after negative numbers and CR after positive numbers. However, you cannot use some functions together, such as D and E.

The $ function displays the expression with the currency symbol immediately before or after the amount. You can only use this function in GETs if SET CURRENCY is LEFT. The currency symbol you want to use, this symbol's placement in the display, the separator characters, and the decimal symbol can be changed with the SET CURRENCY, SET SEPARATOR, and SET POINT commands.

Use S <n> to display and edit a character GET variable. The number of columns you specify, by the literal integer <n>, must be less than the actual length of the character field or memory variable. Do not put any spaces between the S and the integer, and do not include the angle brackets (< >).

The string scrolls within the specified width, allowing you to view and edit the entire character string. Use ←, →, **Home**, and **End** to bring hidden characters into view. When you are entering data, the string scrolls automatically.

The M function allows you to have a list of choices for a GET variable and has the following format:

```
FUNCTION "M <list of choices>"
```

The choices in the list can be either literal strings or literal numbers and must be separated by commas. Since the comma indicates a new choice in the list, do not specify choices with embedded commas. Using two commas at one place in a list creates a blank value.

A value appears in the GET variable only when you move the cursor to the variable position.

GET displays the first item in the list. If the GET variable's value is not the same as one of the values in the list, the variable will contain the first value in the list when you issue a READ. Press the **Spacebar** to see the remaining choices or until the desired value appears. Press ↵ to select an item and move to the next field.

You can also select an item by typing its first letter. If items in the list do not have unique first letters, the next item matching the letter is selected.

The B, I, and J functions trim and align short fields. If you specify no alignment, strings are left-aligned and numbers right-aligned.

## Templates

You form a template by using a single symbol for each character to be displayed or input.

If you use the R function with a template containing characters other than template symbols, those characters are inserted into the display, but not stored as part of the GET variable. If you do not use R, those characters are displayed and are stored in the GET variable. R applies only to character type variables. For numeric variables, non-template symbols are always inserted into the display and never stored as part of the number. Avoid using non-template symbols for date and logical variables.

dBASE IV provides the following template symbols:

Table 2-3 Picture template symbols used in dBASE IV

| Template | Description |
|---|---|
| ! | Converts letters to upper case and has no effect on other characters. |
| # | Allows only digits, blanks, and signs. |
| $ | Displays the current SET CURRENCY string in place of leading zeros. |
| * | Displays asterisks in place of leading zeros. |
| , | Displays if there are digits to the left of the comma. |
| . | Specifies decimal position. |
| 9 | Allows only digits for character data. Allows digits and signs for numeric data. |
| A | Allows only letters. |
| L | Allows only logical data. |
| N | Allows letters and digits. |
| X | Allows any character. |
| Y | Allows only logical Y, y, N, n. Converts y and n to uppercase letters. |

Symbols !, #, 9, A, N, and X may be used with SAYs and GETs. Symbols 9, #, A, N, and X prevent undefined characters from being input, but not from being displayed.

If you use a PICTURE template to GET a decimal number, you must include the decimal point in the template. The template must also leave room for at least one digit to the left of the decimal point, and leave room for the sign, if you want to use the minus sign.

## Programming Notes

If you want to use a multi-page format (.fmt) file in which the @...SAY...GETS continue on from 2 to 32 pages, include a READ wherever you want a page break. The **PgDn** and **PgUp** keys flip the pages. Multi-page format files work only when the .fmt file is opened with SET FORMAT.

## Tip

To design a custom form, use CREATE/MODIFY SCREEN to create a format file (.fmt) that consists of @ commands.

Activate the format file with SET FORMAT TO <.fmt filename>. You can append new records or revise existing ones from an .fmt file using any of the full-screen editing commands such as EDIT and APPEND.

## Examples

To display the information in the first record of the Client database file:

```
. USE Client
. CLEAR
. @ 5,0 SAY TRIM(Firstname) + " " + Lastname
```

As you enter each command, the resulting expression is displayed on your screen. For a more readable display, specify the spaces in quotes as part of the expression, as shown above. The results displayed on your screen should be the following:

```
Fred Wright
```

To provide a list of choices that a user may select in a program file, use the @M PICTURE function. To select a day of the week:

```
SET STATUS ON   && STATUS must be on for the @...GET MESSAGE.
Day_of_wk = "Any"
@ 12,20 SAY "Select the day of the week " GET Day_of_wk;
      PICTURE "@M Mon,Tue,Wed,Thu,Fri,Sat,Sun";
      MESSAGE "Press SPACE to view values and RETURN to select."
READ
```

In this example, "Any" is initially displayed in the input field. As soon as the cursor moves into the field, however, the display changes to "Mon", which is the first item in the list. "Any" is never displayed again since it is not in the list.

Use the VALID clause along with a user-defined function to insure the integrity of input. Continuing with the previous example, enter an amount into a variable called Mrate. Mrate must equal 1, 2, or 3 for a weekday, Saturday, or Sunday, respectively.

```
Mrate = 0
@ 14,20 SAY "The rate is " GET Mrate PICTURE "9";
      VALID Rcheck();
      ERROR "Weekday = 1, Saturday = 2, Sunday = 3"
READ
RETURN

FUNCTION Rcheck
DO CASE
   CASE Day_of_wk <> "S" .AND. Mrate = 1
   RETURN .T.
   CASE Day_of_wk = "Sat" .AND. Mrate = 2
   RETURN .T.
   CASE Day_of_wk = "Sun" .AND. Mrate = 3
   RETURN .T.
ENDCASE
RETURN .F.
```

In the above example, the VALID condition calls the user-defined function Rcheck(). If the validity of the input is correct, Rcheck() returns true (.T.). If Rcheck() is false (.F.), the error message **Weekday = 1, Saturday = 2, Sunday = 3** appears and the user is not allowed to exit the field until the error is corrected.

## See Also

?/??, ACTIVATE WINDOW, APPEND, CHANGE, COL(), CREATE/MODIFY SCREEN, EDIT, INSERT, MODIFY COMMAND, PCOL(), PROW(), READ, ROW(), SET COLOR, SET CONFIRM, SET CURRENCY, SET DELIMITERS, SET DEVICE, SET FIELDS, SET FORMAT, SET INTENSITY, SET POINT, SET SEPARATOR, SET WINDOW OF MEMO, TRANSFORM()

# @...CLEAR

@...CLEAR clears a portion of the screen or of the active window.

## Syntax

@ <row1>, <col1> CLEAR [TO <row2>, <col2>]

## Usage

<row1>, <col1> are the coordinates of the upper left corner of the area that you want to clear, and <row2>, <col2> are the coordinates of the lower right corner.

This command erases the area of the screen starting at <row1>, <col1> up to and including <row2>, <col2>. If you omit the CLEAR TO <row2>, <col2> phrase, the line beginning with <row1>, <col1> is cleared to the end of the line. If you omit TO <row2>, <col2> only, but specify the CLEAR keyword, the screen or active window is cleared from <row1>, <col1> to the bottom right corner.

## Example

To clear the area of the screen from coordinates 2,9 to 14,39:

```
. @ 2,9 CLEAR TO 14,39
```

## See Also

@...FILL, CLEAR

# @...FILL

@...FILL allows you to change the colors of a specific rectangular region on your screen or active window.

## Syntax

@ <row1>,<col1> FILL TO <row2>,<col2>
    [COLOR <color attribute>]

## Usage

This command changes the color of the text in the defined region. <row1>,<col1> are the coordinates of the upper left corner of the region, and <row2>,<col2> are the coordinates of the lower right corner.

In place of <color attribute>, you must provide color codes for the region. These are the same codes used by SET COLOR.

You may change the standard foreground and background colors in the area only. This command affects the display already on the screen. Subsequent commands that write to this area will use the default screen colors, not the colors set with @...FILL.

If you omit the COLOR option, @...FILL clears the rectangular region of the screen and is equivalent to @...CLEAR.

If you specify coordinates larger than your screen, the **Coordinates are off the screen** message appears.

## Example

To paint the screen from coordinates 3,10 to 20,70 in red on black and see the text in that region change color, first issue a LIST MEMORY command to fill the screen with text:

```
. LIST MEMORY
. @ 3,10 FILL TO 20,70 COLOR R/N
```

## See Also

SET COLOR

# @...TO

@...TO draws a box on the screen or active window with single lines, double lines, or specified characters.

## Syntax

@ <row1>,<col1> TO <row2>,<col2>
    [DOUBLE/PANEL/<border definition string>]
    [COLOR <color attribute>]

## Defaults

The default border is a single line, unless it has been changed by the SET BORDER command.

The default color is the NORMAL color, which can also be changed by specifying the NORMAL keyword of the SET COLOR command.

## Usage

<row1>,<col1> are the coordinates of the upper left corner of the box, and <row2>,<col2> are the coordinates of the lower right corner.

If the row coordinates are the same, a horizontal line is drawn. If the column coordinates are the same, a vertical line is drawn.

Defining a border with the @...TO command options overrides the SET BORDER default setting.

## Options

DOUBLE draws a double-line box rather than the default single-line one.

PANEL displays a solid highlighted border. The entire rectangular border is in inverse video.

<border definition string> is a list of character strings (or numbers) used to define a border. The character strings must be delimited, must be separated by commas, and must appear in the following order:

    t,b,l,r,tl,tr,bl,br

The letters stand for the following attributes:

| | | | |
|---|---|---|---|
| t | top | tl | top left corner |
| b | bottom | tr | top right corner |
| l | left | bl | bottom left corner |
| r | right | br | bottom right corner |

If you specify only the first attribute (t), the remaining attributes default to the same value.

You omit an attribute by using a comma in its place if it comes at the beginning of the list, or by simply omitting it if it comes at the end. Omitting an attribute leaves it unchanged from its previous setting.

If you use numbers instead of character strings, use the decimal value of the character in the IBM Extended Character Set. Note that the ASCII code values (decimal 0 through 127) are a subset of this set (decimal 0 through 255). You may also enter numbers as the argument of the CHR() function. The numbers should not be delimited.

COLOR — In place of <color attribute>, you must provide color codes for either foreground, background, or both. These are the same codes used by SET COLOR. If you used the PANEL option, the window will be drawn in the foreground color only. If you do not provide color codes, this command uses the NORMAL colors of the SET COLOR command.

## Programming Notes

You can use the @...TO command from the dot prompt or in a command or format file.

In order to not overwrite the box with a field that is wider than the window, use the S<n> function of the @ command to limit the size of the input area on the screen. The S<n> function allows horizontal scrolling within the input area.

## Example

To draw a double line box from screen coordinates 1,10 to 15,40 with color attributes of black on cyan:

```
. @ 1,10 TO 15,40 DOUBLE COLOR N/BG
```

## See Also

@, @...CLEAR, @...FILL, CHR(), SET BORDER, SET COLOR

# ACCEPT

ACCEPT is used primarily in command files to prompt a user for keyboard entry. It creates a character memory variable in which it stores the keyboard entry. Terminate data entry with ↵.

## Syntax

ACCEPT [ <prompt> ] TO <memvar>

## Usage

The <prompt> may be a character type memory variable, a character string, or any valid character expression. If it is a character string, it must be delimited by single quotes (''), double quotes (""), or square brackets ([ ]).

The keyboard entry does not require delimiters: ACCEPT treats all user input as character-type data.

If ↵ is entered in response to the ACCEPT command, the content of the memory variable is null (without any contents, or ASCII 0).

A maximum of 254 characters can be entered into a variable with ACCEPT.

## Programming Note

Unless SET ESCAPE is OFF, pressing **Esc** in response to an ACCEPT will terminate a program.

## Example

To prompt the user to "Enter your social security number:" and store the keyboard entry in the Ssno memory variable:

```
. ACCEPT "Enter your social security number:" TO Ssno
Enter your social security number:
```

## See Also

@, INPUT, READ, SET ESCAPE, WAIT

# ACTIVATE MENU

This command activates an existing bar menu and displays it for use.

## Syntax

ACTIVATE MENU <menu name> [PAD <pad name>]

## Usage

This command activates a previously defined menu and displays it on the screen over any existing display. If you use the pad name with the ACTIVATE MENU command, the highlight bar appears in that pad. Otherwise, the first pad defined is highlighted.

The last activated menu is the only active menu. Use the → and the ← keys to move between the menu pads. You access the pads in the order in which they were defined. An active menu is deactivated by activating another menu, by pressing **Esc**, or by the DEACTIVATE MENU command.

When one menu activates another, the first menu is suspended until the second is deactivated.

## Example

```
. ACTIVATE MENU Main PAD View
```

Activates the menu called Main and places the cursor in the first pad called View.

## See Also

DEACTIVATE MENU, DEFINE MENU, MENU(), ON PAD, PAD(), SHOW MENU

# ACTIVATE POPUP

This command activates a previously defined popup for use.

## Syntax

ACTIVATE POPUP <popup name>

## Usage

Only one pop-up menu can be active at one time. If you have an active pop-up menu, it is deactivated when you issue a subsequent ACTIVATE POPUP command, press **Esc**, or use the DEACTIVATE POPUP command.

When one pop-up menu activates another, the first pop-up menu is suspended until the second is deactivated.

## Example

This command activates a previously defined pop-up menu which displays some exit options:

```
. ACTIVATE POPUP Exit_pop
```

## See Also

BAR(), DEACTIVATE POPUP, DEFINE POPUP, POPUP(), PROMPT(), SHOW POPUP

# ACTIVATE SCREEN

ACTIVATE SCREEN restores access to the entire CRT screen, rather than to the limits of the recently active window. The most recently active window remains on the screen, but it can be overprinted, scrolled up, or cleared.

## Syntax

ACTIVATE SCREEN

## Usage

ACTIVATE SCREEN sends output to the full screen display instead of to the active window. You do not lose the active window's image, nor do you need to define a window equal to the coordinates of the full screen.

You can use ACTIVATE SCREEN to keep a window's image available for reference while working with full-screen code. Pop-up menus also remain in their correct relative positions.

ACTIVATE SCREEN is also used to place text outside a window, as when an extra message might be needed for a user. In this case, you use ACTIVATE WINDOW after temporarily using ACTIVATE SCREEN to redirect screen output.

Although the active window remains in the foreground, its content is not being updated. The output it would be delivering has been redirected to the full screen, including the area occupied by the window. When output is being sent to the full screen, text can overwrite the window or cause it to scroll off the screen. The CLEAR command can completely clear the screen.

The window remains on the screen until deactivated. If you then reactivate the window, it will cover any full screen text that occupies its position.

## See Also

ACTIVATE WINDOW, DEACTIVATE WINDOW, DEFINE WINDOW

# ACTIVATE WINDOW

The ACTIVATE WINDOW command activates and displays a defined window from memory, and directs all screen output to that window.

## Syntax

ACTIVATE WINDOW <window name list> /ALL

## Usage

To use the ACTIVATE WINDOW command, you must have at least one defined window in memory. Several windows can be present on the screen, but only one window can be active; therefore, if you provide a list of window names, the last window in the list is the active one.

When you use the ALL option, all defined windows currently in memory are displayed in the order they were defined. The borders around each window use the format you established when you defined the window. If you do not specify a border string when you define the window, then the SET BORDER setting determines the borders of the window.

## See Also

CLEAR WINDOW, DEACTIVATE WINDOW, DEFINE WINDOW, MOVE WINDOW, RESTORE WINDOW, SAVE WINDOW, SET BORDER

# APPEND

APPEND allows you to add new records to the end of the active database file.

## Syntax

APPEND [BLANK]

## Usage

APPEND places you in the full-screen data entry mode. One new record at a time is presented in a screen *form* for data entry. If no format has been specified, a default form is displayed.

You terminate the process by pressing ↵ immediately when a new record is presented, or by pressing **Ctrl-End**. If you move to a new record and press **Ctrl-End**, a blank record is added to the file.

The **PgUp** key moves the cursor to previous records, enabling you to edit them. They will be in record order, not indexed order. The **PgDn** key moves forward through the records, and returns to the APPEND mode if you move beyond the last record.

You can move back and forth between the APPEND screen and its menu bar by pressing **F10**.

To enter a memo field, position the cursor on the word *memo* and press **Ctrl-Home**. You leave the field by pressing **Ctl-End**. If you are using dBASE IV's editor to work in a memo field, you use the same control keys as those for MODIFY COMMAND.

APPEND allows you to add records to a single database file only. Using a format file, it is possible to @...GET fields from several related files. Using APPEND with a format file like this does not add records to unselected files. You may, however, use the READ command with the format file to add information to records in several files simultaneously. To add a record to the ends of several files, you must first APPEND a BLANK record to the files, then READ.

## Options

The BLANK option adds a blank record to the end of the database file, but the full-screen mode is not entered. The BLANK record becomes the current record.

## Tips

All active indexes (including .mdx tags) are updated as records are appended.

If SET AUTOSAVE is OFF, the directory entry for the active database file may not reflect all new records until the file is closed. If SET AUTOSAVE is ON, the directory on disk is updated after each new record is added.

## Examples

To enter the full-screen data entry mode and begin APPENDing records to the Client database file:

```
. USE Client
. APPEND
```

To add one blank record to the Client database file without entering the full-screen data entry mode:

```
. APPEND BLANK
```

## See Also

BROWSE, EDIT, SET AUTOSAVE, SET CARRY, SET FORMAT, SET WINDOW OF MEMO

# APPEND FROM

APPEND FROM copies records from an existing file to the end of the active database file. The FROM file does not have to be a dBASE IV file.

## Syntax

```
APPEND FROM <filename>/?
    [[TYPE] <file type>] [FOR <condition>]
```

## Defaults

The filename must include the drive designator if the file is not on the default drive, unless a path is set to that drive.

If you do not provide a file extension as part of the filename, dBASE IV assumes a .dbf extension.

If you do not provide a file extension as part of the filename, but specify SDF or DELIMITED, dBASE IV assumes a .txt extension. dBASE IV also assumes that other file types have the default extensions supplied by their respective software packages.

## Usage

If the FROM file is a dBASE IV database (.dbf) file:

- Records marked for deletion *are* appended and are not marked for deletion in the active database if SET DELETED is OFF. If SET DELETED is ON, only records not marked for deletion are appended.
- Only field names found in both files are appended. They do not have to be in the same order.
- Do *not* specify a file type.

If a field in the FROM file is larger than the same field in the active database, excess character data is lost, and excess numeric data is replaced by asterisks. If the file being APPENDed to has an open index file, the index is automatically updated.

## Options

The options for < file type > are:

- DBASEII — dBASE II database file.
- DELIMITED — Delimited Format ASCII file. Data is appended character by character starting on the left. Each record must end with a carriage return and line feed. A comma separates each field and, in addition, double quotation marks surround character data unless you specify another delimiter. This is the same as DELIMITED WITH ".
- DELIMITED WITH BLANK works with files containing fields separated by one space. No commas separate the fields, and each record ends with a carriage return and line feed.
- DELIMITED WITH < delimiter > works with files containing fields separated by commas and character strings also enclosed in specified delimiters. The default delimiter is the double quote character.
- DIF — VisiCalc file format. The VisiCalc rows convert to records, and the columns convert to fields.
- FW2 — Framework II™ database or spreadsheet frame.
- RPD — RapidFile™ data file.
- SDF — System Data Format ASCII file. Also known as a *fixed-length* file. Data is appended character by character starting on the left. Each record in the FROM file is the same length and ends with a carriage return and line feed, and individual fields are not delimited.
- SYLK — MultiPlan spreadsheet format in *row major* order. The MultiPlan rows convert to records, and the columns convert to fields. dBASE IV will not APPEND FROM a SYLK file saved in *column major* order.
- WKS — Lotus 1-2-3 spreadsheet format, release 1A. The Lotus 1-2-3 rows convert to records, and the columns convert to fields. The file begins with the cell in the upper left-hand corner of the spreadsheet.

  Blank rows in any spreadsheet type are converted to blank records in the database file.

  When using APPEND FROM to read any of the supported spreadsheet formats, keep in mind that this command expects the incoming data in a format that matches the database file structure. This means that you should have stored the spreadsheet in *row major* (as opposed to *column major*) order, and that you should remove column headers before attempting to read the data into dBASE IV. However, APPEND FROM will store row names as long as the database file structure is designed with them in mind. Lotus 1-2-3 files should not have leading blank rows or columns. With this type of file, you should justify data in the upper left-hand corner before using APPEND FROM.

### Tip

If you are APPENDing FROM a file for which dBASE IV assumes a particular extension and your file does not have an extension, include a period after the filename.

### Special Cases

Use the IMPORT command to convert PFS:FILE and Lotus release 2.x formats to dBASE IV format. You may also use IMPORT to convert dBASE II, Framework II, and RapidFile files to dBASE IV files.

dBASE IV date fields can APPEND FROM character fields, if the data is in the proper date format. Conversely, character fields in the proper format and size can APPEND FROM date fields. From text files, dates can only be APPENDed FROM in the form YYYYMMDD, not delimited (where YYYY is the year, MM is the month, and DD is the day).

### Example

To APPEND FROM a RapidFile database named Contacts into the Clients database file:

```
. USE Client INDEX Cus_name
. APPEND FROM Contacts.rpd TYPE RPD
```

This example opens Cus_name.ndx to ensure the integrity of the index. Otherwise, Cus_name.ndx would have to be REINDEXed the next time you open it.

### See Also

COPY, DELETED(), IMPORT, SET DELETED

# APPEND FROM ARRAY

APPEND FROM ARRAY adds records to a database file from information in an array.

## Syntax

APPEND FROM ARRAY <array name> [FOR <condition>]

## Usage

The contents of each row in the named array may become a new record in the current database file.

For each row in the array, the contents of the first column are replaced into the first field, the contents of the second column are replaced in to the second field, and so on. This process continues until there are either no more array columns or no more fields.

If an array has more columns than the database has fields, the excess array columns are ignored. If the database file has more fields that the array has columns, the excess fields remain empty.

Each element in the array must be the same data type as the field into which it will be replaced.

If you use the FOR clause, the condition is evaluated before each row in the array is added to the database file. The <condition> in the FOR clause must include a fieldname from a database file. A row is APPENDed only if the condition evaluates to true (.T.). If the condition is false (.F.), the row is skipped and the next row is processed.

You must DECLARE the array and STORE information to its elements before you can APPEND from it.

## Example

To APPEND FROM an ARRAY into the Transact database file, first establish the array:

```
. DECLARE Newdata[1,5]
. Newdata[1,1] = "M00001"
M00001
. Newdata[1,2] = "88-100"
88-100
. Newdata[1,3] = DATE()
03/01/87
. Newdata[1,4] = .F.
.F.
. Newdata[1,5] = 125.00
       125.00
. USE Transact
. APPEND FROM ARRAY Newdata
   1 record added
```

## See Also

APPEND, COPY TO ARRAY, DECLARE, STORE

# APPEND MEMO

APPEND MEMO imports a file into a named memo field.

## Syntax

APPEND MEMO <memo field name> FROM <filename> [OVERWRITE]

## Defaults

If you do not specify a filename extension, the default extension .txt is assigned.

## Usage

APPEND MEMO can read any file into a memo field. The entire file is read and added to the existing contents of the memo field.

Use the OVERWRITE option to erase the existing contents before adding new content to the memo field.

If the named field cannot be found, the message **Field not found** appears.

If the named field is not a memo field, the message **Field must be a memo field** appears.

If the specified filename cannot be found, the message **File does not exist** appears.

## Examples

To APPEND a text file named Newtext.txt to the first record in the Client database file, first create the text file with the MODIFY FILE command:

```
. MODIFY FILE Newfile.txt
New text begins here...
```

Save the file with **Ctrl-End**. Then, from the dot prompt:

```
. USE Client
. ? Clien_hist
85-200 08/02/85
   C-300-400 BOOK CASE          535.00 1
. APPEND MEMO Clien_hist FROM Newfile
. ? Clien_hist
85-200 08/02/85
   C-300-400 BOOK CASE          535.00 1
New text begins here...
```

## See Also

COPY MEMO, MODIFY COMMAND/FILE

# ASSIST

ASSIST gives you access to dBASE IV's Control Center.

## Syntax

ASSIST

## Usage

ASSIST brings you to the Control Center, from which you can reach the different parts of dBASE IV's Menu System.

The Control Center always opens a catalog. It first attempts to open the most recently opened catalog in the master catalog. If a catalog is not available, it creates a new catalog called Untitled.cat.

To begin using the Menu System, follow the instructions on the screen.

The Control Center is a gateway to six design screens, each represented by a panel in the Control Center. These design screens can also be reached using the CREATE, CREATE/MODIFY QUERY/VIEW, CREATE/MODIFY SCREEN, CREATE/MODIFY REPORT, CREATE/MODIFY LABEL, and CREATE/MODIFY APPLICATION commands.

You may also view, edit, and add data, as with BROWSE, EDIT, and APPEND; run reports and labels, as with REPORT FORM and LABEL FORM; execute programs and other files, as with DO, SET VIEW, and SET FORMAT; and enter the program editor, as with MODIFY COMMAND/FILE.

To leave the Control Center and return to the dot prompt, press **Esc**; or, you can press **Alt-E** to open the **Exit** menu, then select the **Exit to dot prompt** option.

## See Also

The Control Center is described in *Learning dBASE IV* and in *Using the Menu System*. Please refer to these manuals for more information.

# AVERAGE

AVERAGE computes the arithmetic mean of numeric expressions.

## Syntax

AVERAGE [ < expN list > ] [ < scope > ] [FOR < condition > ]
  [WHILE < condition > ] [TO < memvar list > /TO ARRAY
  < array name > ]

## Defaults

All records are averaged unless otherwise specified by the scope or the FOR or WHILE clause. All numeric fields are averaged unless otherwise specified by an expression list. The result of AVERAGE is displayed on the screen as long as SET TALK is ON.

## Usage

If you use the TO ARRAY phrase, the array must be one-dimensional. The results are stored in the named array, in order beginning with the first slot. If there are more results than the original array declaration, the excess results are not stored. If there are fewer results, the remaining array elements remain unchanged.

## Example

To obtain the average Total_bill for Client_id C00002 in the Transact database file:

```
. USE Transact
. AVERAGE Total_bill FOR Client_id = "C00002"
     2 records averaged
          Total_bill
              712.50
```

## See Also

CALCULATE, DECLARE, SET HEADING, SET TALK, SUM

# BEGIN/END TRANSACTION

This two-phase command starts a transaction, records changes made to a database file, and provides the option to ROLLBACK those changes. The second phase of the command commits the changes and removes the option to ROLLBACK. Both phases of the command are always used together.

## Syntax

```
BEGIN TRANSACTION [ <path name> ]
<transaction commands>
END TRANSACTION
```

## Usage

You can use this commmand in a program or from the dot prompt to start a transaction recording of the changes you make to records in a database file. Each transaction procedure begins with BEGIN TRANSACTION and ends with END TRANSACTION.

A process that modifies a single record in a database is referred to as an *atomic* action. Atomic actions are:

- Adding new records
- Changing the data in records
- Changing the delete status of records

BEGIN TRANSACTION starts recording these atomic changes in a transaction log file which it creates. The transaction log file is named Translog.log on a standalone system, and <Workstation name>.log on a local area network. <Workstation name> is the name of the computer that starts the BEGIN TRANSACTION command on the network. Specify a path name if you want the log file in a specific directory; otherwise, it will be created in the current directory.

In addition to the transaction log file, BEGIN TRANSACTION sets the integrity tag located in the database file header to indicate that the file is in a state of change.

If you decide to ROLLBACK from the changes you made, then this log file is read to restore the database file to its unchanged state. While a transaction is active, before you issue an END TRANSACTION command, you can undo all unwanted changes. Use the ROLLBACK command to restore the database file to its pre-transaction state.

If the change is satisfactory, issue an END TRANSACTION command. This command terminates the active transaction, deletes the transaction log file, and clears the integrity tag from the database file header.

You should not use this command until you are certain that the transaction has done what you wanted it to do. Once you issue an END TRANSACTION command and the transaction log file is erased, the only way to restore the database file to its pre-transaction state is to use backup files.

A transaction must be complete and the log file deleted before you can start another transaction; transactions cannot be nested. To perform several transactions in succession, bracket each transaction within a BEGIN TRANSACTION and an END TRANSACTION command. Both BEGIN TRANSACTION and END TRANSACTION must be contained in the same procedure file.

## Transaction Recording

A group of dBASE IV commands that change database files or the operating environment are recorded in the Transaction.log file. These commands cause atomic changes, update catalogs, or create new database or index files. dBASE IV commands that create new files are allowed in transactions *unless these commands overwrite existing files* or close open files. Overwriting existing files or closing open files is not allowed, as this would make ROLLBACK impossible.

dBASE IV commands that cause atomic actions to database files are:

```
APPEND
BROWSE
CHANGE
DELETE
EDIT
RECALL
REPLACE
UPDATE
```

dBASE IV commands that create new files are allowed during transaction processing if they do not close open files. These are:

```
COPY [STRUCTURE] TO <newfile> [EXTENDED]
CREATE <newfile> [FROM <filename>]
EXPORT TO <filename>
IMPORT FROM <filename>
INDEX...TO <newfile>
JOIN...TO <newfile>
SET CATALOG TO <newfile>
SORT...TO <newfile>
TOTAL...TO <newfile>
```

Some of the same commands that are allowed when they create new files are not allowed during a transaction if they close open files.

These commands are:

CREATE [FROM]
IMPORT
INDEX ON
SET CATALOG TO
SET INDEX TO
USE

### Commands Not Allowed in Transactions

The following dBASE IV commands are never allowed during a transaction:

CLEAR ALL
CLOSE ALL/DATABASE/INDEX
DELETE FILE
ERASE
INSERT
MODIFY STRUCTURE
PACK
RENAME
ZAP

Attempting to use any of these commands during a transaction results in an error message.

The UNLOCK command is a special case; while it does not produce an error message, it has no effect during a transaction.

## Tip

Back up database files that are going to be involved in transactions. In case you do not like the changes you made, and the ROLLBACK command is not successful in restoring the files to their beginning state, you can use the backup copies to undo the transaction manually.

## Examples

The following routines, Invoice and Err_proc, are examples of transaction processing with error checking and handling:

```
PROCEDURE Invoice
SET REPROCESS TO 15
ON ERROR DO Err_proc
BEGIN TRANSACTION
   USE Invoice ORDER Acct_no
```

```
      SCAN FOR .NOT. Invoiced
         *
         *
         *
       ENDSCAN
   END TRANSACTION
   IF .NOT. ROLLBACK()
      @ 21, 15 SAY "The rollback was not successful.;
      You must restore"
      @ 22,15 SAY "from the backup before continuing."
   ENDIF
   ON ERROR
   RETURN

   PROCEDURE Err_proc
   choice = " "
   DO CASE
      CASE ERROR() = 108     && File in use by another
         @ 21,15 SAY "One of the files that you need;
         is in use by another."
         @ 22,15 SAY "Do you want to try to use it again? (Y/N)";
         GET choice
         READ
         @ 21, 15 CLEAR TO 22,65
         IF UPPER (choice) = "Y"
            RETRY
            RETURN
         ELSE
            IF COMPLETED()
               *--- Error did not occur during a transaction.
               @ 21,15 SAY "The file in use by another is;
               not part of a transaction."
               @ 22, 15 SAY "Returning to the main menu."
               RETURN TO MASTER
            ENDIF
            @ 21, 15 SAY "Rolling back your entries. Please wait..."
            ROLLBACK
         ENDIF
         RETURN TO MASTER
      CASE ERROR() = 109
         *** Similar type of routine.
   ENDCASE
   RETURN
```

## See Also

COMPLETED(), ISMARKED(), RESET, ROLLBACK, ROLLBACK()

# BROWSE

BROWSE is a full-screen, menu-assisted command for editing and appending records in database (.dbf) files and views.

## Syntax

```
BROWSE [NOINIT] [NOFOLLOW] [NOAPPEND] [NOMENU] [NOEDIT]
    [NODELETE] [NOCLEAR] [COMPRESS] [FORMAT] [LOCK <expN>]
    [WIDTH <expN>] [FREEZE <field name>][WINDOW
        <window name>]
    [FIELDS <field name 1> [/R] [/<column width>]
        / <calculated field name 1> = <expression 1>
        [,<field name 2> [/R] [/<column width>]
        / <calculated field name 2> = <expression 2>] ...]
```

## Defaults

If a database file or view is not in USE before you BROWSE, the command will prompt you to enter a database filename.

Calculated fields are always read-only.

## Usage

BROWSE displays records from database files or views in tabular form. All fields are displayed in the order specified by the file structure or view definition, or in the order listed after the FIELDS option. If the files have been PROTECTed, the fields must also have a read/write or read-only attribute.

You can change from BROWSE to EDIT by pressing the **F2 Data** key.

You can edit all fields except calculated fields or read-only fields. When you move the cursor to a field that can be edited, the field appears in enhanced video. The cursor also indicates where you can make changes in the field. If the changes you make affect any calculated fields, the calculated fields are recalculated and redisplayed.

If an index file is active, editing a key field value repositions the record according to its new key value in the index.

You can also add records to the active file or view. Move the cursor to the bottom of the file and press ↓. A prompt asks if you want to add records to the file. If you answer Y, BROWSE goes to APPEND mode and allows you to add a record to the file. You can also add records to a currently selected database file contained in a view.

Press **F10** to activate the BROWSE menu bar. The BROWSE menus are described fully in *Learning dBASE IV* and *Using the Menu System*. Please refer to these manuals for more information.

If you call BROWSE from the dot prompt, you return to the dot prompt when you exit BROWSE. In .prg files, you return to the next statement after the BROWSE command.

## Options

NOINIT allows the command line options that you used with a previous BROWSE command to be used in the current EDIT. NOINIT instructs the BROWSE command not to initialize the BROWSE table, but to use the table from the most recent BROWSE instead.

NOFOLLOW affects only indexed files. Ordinarily, editing a key field value repositions the record according to its value in the index, and the record remains the current one. If you issue NOFOLLOW before changing the field's contents, the record is repositioned, but the record that took its original place in the file order becomes the current record.

NOAPPEND keeps you from adding new records to the current file when in BROWSE mode.

NOMENU prevents access to the menu bar and keeps it from being displayed.

NOEDIT makes all fields in the table read-only. Using this option means that no records can be edited in BROWSE mode. You can still add records to the file. You can also mark records for deletion.

NODELETE prevents you from deleting records by pressing **Ctrl-U** when in BROWSE mode.

NOCLEAR leaves the BROWSE table on the screen after you exit BROWSE.

COMPRESS slightly reduces the table format to allow two more lines of data to appear on the screen. The column headings are placed on the top line of the table border, and no line separates the headings from the data. On a 25-line screen, BROWSE presents up to 17 records if you don't use the COMPRESS option, but up to 19 records with COMPRESS.

FORMAT instructs BROWSE to use the @...GET command options specified in the active format (.fmt) file. All @...GET command options except the positioning of fields on the screen, which is controlled by the @ command's row and column coordinates, are then used by BROWSE. Row and column coordinates are ignored, because BROWSE always positions fields in a table on the screen.

LOCK specifies the number of contiguous fields on the left of the screen that do not move when you are scrolling the BROWSE table. When you press **F3** or **F4** to scroll fields left or right, the number of fields you *locked* will remain displayed in the same position on the screen.

LOCK 0 will undo all locked fields.

WIDTH sets an upper limit on the column widths for all fields in the BROWSE table. The specified WIDTH overrides the width defined by the database structure. If both WIDTH and the <column width> options are used for a field, the smaller value of the two will take precedence.

The WIDTH option does not apply to memo fields or logical fields. Numeric and date fields will not display in a column WIDTH that is less than the width of these fields in the database file structure.

FREEZE confines you to one field or column in the file. If the field has been made read-only, you may not edit it. Other fields in the field list or database file table can display on the screen if they fit, but aren't subject to editing.

FREEZE without a field name will undo a previously established FREEZE.

WINDOW activates a window which then defines an area on the screen used by the BROWSE table. The rest of the screen remains intact. When you exit BROWSE, the window is automatically deactivated, unless you also used the NOCLEAR option. If a window is active when BROWSE is called, BROWSE displays data in that window.

FIELDS allows you to choose fields and specify the order in which they appear in the table. If the file or view is PROTECTed and you do not have access to a field, it is not displayed. The FIELDS option also allows you to designate a field as read-only, and to construct and name calculated fields that will appear in the BROWSE table.

Each calculated field is composed of an assigned field name and an expression that results in the calculated field value, as *Commission = Rate*Saleprice*.

The field name you assign becomes the column header for the calculated field in the BROWSE table. dBASE IV determines the length of a calculated field after evaluating the expression in the first record.

The /R option makes a field read-only. The read-only flag set by PROTECT has precedence over this option. Calculated fields don't need /R protection.

The <column width> option is a numeric literal that defines the column width of a field and can be used with each field in the fields list. It must be preceded by a slash (/) in the command line.

<column width> can be a value from 4 to 100 for character fields, and from 8 to 100 for date and numeric fields. <column width> is ignored for memo fields and logical fields. It is also ignored if it is larger than the WIDTH option, or if the value is less than the database file structure's width for date or numeric fields.

## Tip

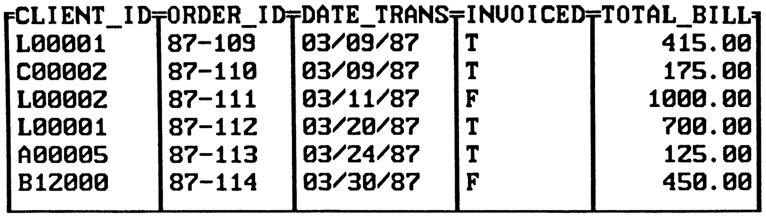
If SET AUTOSAVE is OFF, the directory entry for the active database file may not reflect all new records until the file is closed. If SET AUTOSAVE is ON, the directory on disk is updated after each new record is added.

## Examples

In a program file use BROWSE to display up to seven records from the Transact database file. Limit the size of the display to seven records by declaring a window called Partial. Filter the database file for orders that occurred in March. Allow the user to move within the BROWSE table, but not to make changes to the database file. Finally, leave the BROWSE table on the screen while another routine utilizes the lower portion of the screen. Leaving the table on the screen allows the user to see the data.

```
USE Transact
SET FILTER TO MONTH(Date_trans) = 3   && Filter for March.
GO TOP                                && Against the filter.
DEFINE WINDOW Partial FROM 2,10 TO 9,60
BROWSE NOAPPEND NOEDIT NODELETE NOCLEAR NOMENU COMPRESS WINDOW Partial
DO Next_prg
```

The BROWSE command in the example creates a display like the following:



```
CLIENT_ID ORDER_ID DATE_TRANS INVOICED TOTAL_BILL
L00001    87-109   03/09/87   T            415.00
C00002    87-110   03/09/87   T            175.00
L00002    87-111   03/11/87   F           1000.00
L00001    87-112   03/20/87   T            700.00
A00005    87-113   03/24/87   T            125.00
B12000    87-114   03/30/87   F            450.00
```

Figure 2-1 BROWSE window

## See Also

APPEND, EDIT, PROTECT, SET MEMOWIDTH, SET REFRESH, SET WINDOW OF MEMO

# CALCULATE

CALCULATE computes financial and statistical functions with your data.

## Syntax

```
CALCULATE [scope] <option list>
    [FOR <condition>] [WHILE <condition>]
    [TO <memvar list>/TO ARRAY <array name>]
```

where <option list> can be any one of the following functions:

```
    AVG(<expN>)
    CNT()
    MAX(<exp>)
    MIN(<exp>)
    NPV(<rate>,<flows>,<initial>)
    STD(<expN>)
    SUM(<expN>)
    VAR(<expN>)
```

## Default

All records in the active file are processed if a scope is not defined.

## Usage

The CALCULATE command handles all records in the active database file until the scope is completed, or the FOR or WHILE condition is no longer true. The result of the command is one or more type N numbers.

If you include a FOR clause, each record is evaluated and, if true (.T.), the specified functions are evaluated and performed.

If you include the TO ARRAY option, the results are stored in the named array which must be one-dimensional. The array slots are filled in order beginning with the first one.

If SET TALK is ON, the results appear on the screen. If SET HEADING is ON, the results are also labeled with the name of the function and the field name.

## Options

The financial and statistical optional functions are described below. You can save the results of these options with the TO option.

**NOTE**
*Records with blank values are used in calculating the maximum, and are treated as fields that are equal to zero.*

AVG( < expN > ) calculates the arithmetic mean of the values in a particular database field. It returns a number that is the same data type as the argument.

CNT() counts the records in a database file. The FOR condition is evaluated for each record. If the condition is true (.T.), the record count is increased by one.

MAX( < exp > ) determines the largest number in a particular database field. < exp > is a numeric, date, or character expression that will normally be the name of a field, or an expression involving the name of a field.

MIN( < exp > ) determines the minimum number in a particular database field. Use it in the same manner you would use MAX(), except you will determine a minimum value.

NPV( < rate > , < flows > , < initial > ) calculates the net present value of the specified database field.

< rate > is the discount rate represented by a decimal number.

< flows > is a series of signed ( + / – ) periodic cash flow values. If < initial > is not used, the series begins with the present period. < flows > can be any valid numeric expression, but will normally be the name of a field, or an expression involving the name of a field.

< initial > is a numeric expression whose value represents an initial investment. The initial value should be a negative number since it represents cash outflow. The output of NPV() can be type N (fixed) or type F (floating), depending on the arguments you use.

STD( < expN > ) calculates the standard deviation of the values in a database field. The formula for the standard variation is the square root of the variance. < expN > is a numeric expression that will normally be the name of a field, or an expression involving the name of a field. The output of STD() can be type N (fixed) or type F (floating), depending on the arguments you use.

SUM( < expN > ) determines the sum of the values in a particular database field. The FOR condition is evaluated for each record. If the condition is true, the value for that record is added to the previous total. < expN > is a numeric expression that will normally be the name of a field, or an expression involving the name of a field.

VAR( < expN > ) calculates the population variance of the values in a particular database filed. < expN > is a numeric expression that will normally be the name of a field, or an expression involving the name of a field. The output of VAR() is a Type F number.

## Examples

To find the sum of all invoiced orders in the Transact database file:

```
. USE Transact
. CALCULATE FOR Invoiced SUM(Total_bill), CNT() TO total, cnt
     SUM(Total_bill) CNT()
        6965.00          8.00
. ? "The "+LTRIM(STR(cnt,3))+" invoiced orders total $"+LTRIM(STR(total,8,2))
The 8 invoiced orders total $6965.00
```

To calculate the average bill and the standard deviation of the Total_bill field:

```
. USE Transact
. CALCULATE STD(Total_bill), AVG(Total_bill) TO Mdeviate, Maverage
    12 records
     STD(Total_bill)     AVG(Total_bill)
               555.92               840

. Mdeviate = LTRIM(STR(Mdeviate,8,2))
555.92
. Maverage = LTRIM(STR(Maverage,8,2))
840.00
. ? "The average bill is $"+Maverage+" with a deviation of $"+Mdeviate+"."
The average bill is $840.00 with a deviation of $555.92.
```

To find the last date that a transaction took place and the largest order to date:

```
. USE Transact
. CALCULATE MAX(Date_trans), MAX(Total_bill) TO last_trans, largest
     MAX(Date_trans) MAX(Total_bill)
04/10/87         1850.00
. SET CENTURY ON
. ? "The last order was on " + DMY(last_trans) + "."
The last order was on 10 April 1987.
. ? "The largest order was $" + LTRIM(STR(largest,8,2)) + "."
The largest order was $1850.00.
```

The CALL command allows you to call binary file program modules loaded in memory. You must have first loaded the binary program files in memory using the LOAD command.

## Syntax

CALL <module name> [WITH <expression list>]

## Usage

The CALL command executes a binary program file that has been loaded in memory with the LOAD command. dBASE IV treats each loaded file as a subroutine or module rather than as an external program (which could be executed with the RUN command). As a result, each time you want to execute the program, it can simply run from memory without having to be reloaded from a disk. Up to 16 binary program files can be loaded in memory at one time, and each can be up to 32,000 bytes long.

When you CALL the binary program file, you specify the name of the file without the .bin extension. When you call the file, you can pass either a character expression, a memory variable, or an array element of any data type to the binary program file. All character type memory variables and expressions end with a null (ASCII value of zero).

When the program module is called, the Code Segment (CS) points in memory to the beginning of the module. The Data Segment (DS) and BX registers contain the address of the first byte of the memory variable, field, array element, or character expression.

## Options

The WITH option is a character expression, field name, memory variable (or array element). You can use up to seven expressions in the <expression list>. Memory variables, fields, and elements can be of any type. Character expressions are called by value, but all other parameters are called by name.

## Example

See the LOAD command.

## See Also

CALL(), LOAD, RELEASE, RUN/!

# CANCEL

CANCEL stops the execution of a command file, closes all open command files, and returns dBASE IV to the dot prompt. CANCEL does not close procedure files.

## Syntax

CANCEL

## Usage

dBASE IV ignores any text on lines in a dBASE program file following CANCEL.

CANCEL is most often used to assist in the process of debugging programs.

## Example

To stop command processing when an error is found, include CANCEL within an IF...ENDIF or DO CASE...ENDCASE structure. In the following example, the Merror memory variable was previously established:

```
IF Merror                    && If error is true, command
   CANCEL                    && file execution is cancelled.
ENDIF
```

## See Also

RESUME, RETRY, RETURN, SET REPROCESS, SUSPEND

# CHANGE

CHANGE is an alternate syntax for EDIT, a full-screen command you use to display or change the contents of a record in the active database file or view.

## Syntax

```
CHANGE [NOINIT] [NOFOLLOW] [NOAPPEND] [NOMENU] [NOEDIT]
    [NODELETE] [NOCLEAR] [<record number>] [FIELDS <field list>]
    [<scope>] [FOR <condition>] [WHILE <condition>]
```

## Usage

The CHANGE command is identical to the EDIT command. See EDIT for further information about how to use this command.

## See Also

EDIT

# CLEAR

CLEAR erases the screen, repositions the cursor to the lower left-hand corner of the screen, and releases all pending GETs created with the @ command.

Various forms of the CLEAR command also close database files; release memory variables, field lists, windows, popups, and menus; and empty the type-ahead buffer.

## Syntax

```
CLEAR [ALL/FIELDS/GETS/MEMORY/MENUS/POPUPS/TYPEAHEAD
    /WINDOWS]
```

## Usage

You must use the CLEAR command either without any options, or with one option only. You may not use more than one option in a command line.

## Options

ALL closes all open database files; releases all memory variables (except system variables), array elements, popup and menu definitions; and selects work area 1. Closing the database files also closes their associated index (both .ndx and .mdx) files, format (.fmt) files, and memo (.dbt) files. CLEAR ALL also closes the catalog (.cat) file if one is active.

FIELDS releases the fields list created by the SET FIELDS command. CLEAR FIELDS has no effect unless you used SET FIELDS, or the command was activated by a view or query file. Unlike SET FIELDS TO, which removes only the active database fields from the fields list, CLEAR FIELDS releases all fields, including those from other work areas. CLEAR FIELDS automatically performs a SET FIELDS OFF.

GETS releases all @...GETs issued since the last CLEAR ALL, CLEAR GETS, or READ command. Unless the GETS parameter is redefined using a Config.db file, dBASE IV permits 128 @...GETs before a CLEAR GETS or READ must be issued.

The maximum number of GETs allowed is 1,023. However, the number you can actually use is limited by the amount of memory in your computer. (See your *Getting Started* booklet for memory limitations.) For most purposes, the default value of 128 is sufficient.

The READ command releases all GETs, unless you use the SAVE option. If you use the SAVE option, you must CLEAR GETs before the maximum number of GETs is reached.

CLEAR GETS does not release memory variables or array elements.

MEMORY releases all memory variables (except system variables) and array elements. CLEAR MEMORY, when issued at the dot prompt, performs the same function as the RELEASE ALL command. When used in programs, however, CLEAR MEMORY releases all PUBLIC and PRIVATE memory variables and elements, while RELEASE ALL releases only PRIVATE memory variables and elements declared in the program currently being executed.

MENUS clears all user menus from the screen and erases them from memory. Use this command to clear the screen of all menus, and to make the memory used by menus available for other operations.

POPUPS clears all pop-up menus from the screen and erases them from memory. Use this command to clear the screen of all pop-up menus, and release the memory used by pop-up menus. While clearing popups, this command also DEACTIVATEs the active popup, and clears all ON SELECTION commands associated with the pop-up menus.

TYPEAHEAD empties the type-ahead buffer. Use CLEAR TYPEAHEAD when you want to make sure that there are no keystrokes in the type-ahead buffer. This is particularly useful in programs when you want the user to enter information before the program continues. For instance, place it before CHANGE, READ, WAIT, or similar commands to ensure that dBASE IV acts upon the correct characters.

WINDOWS removes all windows from the screen and clears them from memory. Any window definitions that you have not saved before issuing this command are lost. If you save your window definitions to a disk file, you can RESTORE WINDOWS from a disk file whenever you wish, and remove them from the screen and memory quickly with the CLEAR WINDOWS command.

When you issue a CLEAR WINDOWS command, the full screen is restored. Any text that was covered up by the windows becomes visible.

## See Also

@, @...FILL, @...TO, CLOSE, CREATE/MODIFY VIEW, READ, RELEASE, SET FIELDS, SET FORMAT, SET TYPEAHEAD, SET VIEW

Chapter 6, "Customizing dBASE IV"

# CLOSE

CLOSE is used to close alternate files, database files, format files, index (.ndx and .mdx) files, and procedure files.

## Syntax

CLOSE ALL/ALTERNATE/DATABASES/FORMAT/INDEX/PROCEDURE

## Usage

CLOSE ALL closes all files of all types and reselects work area 1.

To CLOSE a particular file type, follow the command with one of the keywords given above.

CLOSE DATABASES does not affect work area 10 if a catalog is open. Besides closing all database files, it closes any associated index (.ndx and .mdx) files, memo (.dbt) files, and format (.fmt) files.

If you want to close only the currently selected database file and its associated files, issue the USE command, rather than the CLOSE command.

## Examples

To close all open files:

```
. CLOSE ALL
```

To close all open database, index, and format files:

```
. CLOSE DATABASES
```

# COMPILE

COMPILE reads a file containing dBASE IV source code, and creates an executable object code file.

## Syntax

COMPILE <filename> [RUNTIME]

## Defaults

Unless the file is on the default drive or a path is set, the filename must include the drive designator. The filename must also include a path, if the file is not on the default directory or on a path set with the SET PATH command.

COMPILE can only compile a source file. This command looks for a file with a .prg extension, unless you provide another extension in the command line. Source files contain dBASE IV commands and functions, and typically have a .prg (program), .prs (SQL program), .fmt (format), .frg (generated report form), .lbg (generated label), .qbe (query), or .upd (update query) extension.

Although the source file may have any extension, COMPILE will always write the object file with a .dbo extension.

## Usage

Object files contain an execute-only form of dBASE code. Earlier versions of dBASE did not generate object code files. COMPILEd files, therefore, cannot be used by dBASE III PLUS or dBASE III. These object files allow dBASE IV to execute much faster than earlier versions.

You cannot modify an object file. You can modify the source code file, but should verify that changes to the source code file are COMPILEd to the object code file.

By including the RUNTIME option, you cause dBASE IV to print out any errors that occur from commands that aren't allowed in RunTime use.

When you DO a program file, the source code is compiled into an object code file, and then the object code is executed. COMPILE allows you to generate the object code file without executing the code.

If SET DEVELOPMENT is ON, DO compares the time and date stamp of a .prg file with the time and date stamp of its associated .dbo file. If the .dbo file is older than the .prg file, DO recompiles the .prg file before executing it.

The dBASE IV program editor, which can be accessed from MODIFY COMMAND or the Control Center, will delete an old .dbo file when a .prg file is modified, and then recompile the new .prg file (generating a new .dbo file) when you save it. dBASE IV will erase a prior .dbo file for a .prg file modified by an external text editor, if the external editor is identified in the Config.db file.

If you do not use the MODIFY COMMAND program editor, and do not SET DEVELOPMENT ON, you must recompile your source code files after modifying them, or else the changes will not be written to the object file.

COMPILE checks each line of the source code file for syntactical accuracy and proper control structure, and flags syntax errors and control structure violations (such as missing ENDDOs and ENDIFs). dBASE IV displays messages about these events during compilation, provided SET TALK is ON. If a macro appears in a command line, however, it is expanded and parsed at execution time.

As COMPILE always writes the object file with a .dbo extension, you should give source code files unique names so that one file does not overwrite a file COMPILEd earlier with the same name.

dBASE IV supports the concept of procedures within any program file by maintaining a procedure list at the beginning of every object (.dbo) file. If a source file starts with a command other than PROCEDURE or FUNCTION, the code is compiled as a procedure and added to the procedure list in the object file with the same name as the source file. A typical .prg file such as:

```
* Main.prg
? "MAIN"
RETURN
```

is compiled into a .dbo file containing one procedure, Main. When you enter DO MAIN, this name is used to locate the .dbo file, and then used to locate the procedure within the .dbo file.

A source file can include more than one procedure, such as:

```
* Main.prg
? "MAIN"
DO Subb
RETURN

PROCEDURE Subb
? "SUBB"
RETURN
```

Note that only code found at the beginning of a file is given the default procedure name.

Any procedure found in an active .dbo file is available to the DO command. If A.dbo calls B.dbo calls C.dbo, all the procedures defined within A, B, and C are available to any procedure in C. dBASE IV still supports SET PROCEDURE from dBASE III and dBASE III PLUS, although this command is only required to gain access to procedures in a file not activated by DO <filename>.

## Tips

COMPILE places all command keywords in a pre-designated order before generating object code. This order is usually the same as given in the paradigms under the "Syntax" heading for each command and function. Compile time can be improved for long commands by entering commands in the order given in the syntax paradigms.

dBASE IV optimizes expressions during compilation. If you use constants in the source code, the compiler computes and saves the value in the object code file. For example,

```
x = 1 + 3 + 4
```

is optimized and saved as

```
x = 8
```

Comparing two string constants in an expression could cause a problem. For example,

```
"abcde" = "abcd"
```

evaluates to true (.T.) if SET EXACT is OFF, but to false (.F.) if SET EXACT is ON. If you SET EXACT ON during execution of the code, the expression will still be false because it was optimized and saved as a logical false (.F.) during compilation.

## See Also

DEBUG, DO, FUNCTION, MODIFY COMMAND, PROCEDURE, SET DEBUG, SET DEVELOPMENT, SET EXACT, SET PATH, SET PROCEDURE, SET TRAP

# CONTINUE

CONTINUE searches for the next record in the active database file that meets the condition specified by the most recent LOCATE command.

## Syntax

CONTINUE

## Usage

A CONTINUE search ends when it finds a record that meets the specified LOCATE condition, or when it reaches the end of the LOCATE *scope* or the end of the file.

LOCATE and CONTINUE are specific to the work area in which you issue them. You can issue different LOCATE and CONTINUE commands in each work area. If you leave a work area, the LOCATE condition will still be in effect when you return.

## Record Pointer

If SET TALK is ON, and if CONTINUE finds another record meeting the condition, the record number is displayed. Otherwise, the message **End of LOCATE scope** appears, FOUND() returns a logical false (.F.), and the record pointer is positioned at the last record of the LOCATE scope or at the end of the file. If the record pointer is at the end of the file, EOF() returns a logical true (.T.).

## Example

To locate records containing "OAK" in the Descript field of the Stock database file:

```
. USE Stock
. LOCATE FOR "OAK" $ Descript
Record =      6
. CONTINUE
Record =     15
. CONTINUE
End of LOCATE scope
```

The message **End of LOCATE scope** indicates that no more records match the FOR or WHILE condition, or the scope of the LOCATE command.

## See Also

EOF(), FIND, FOUND(), LOCATE, SEEK, SEEK()

# CONVERT

CONVERT adds a field to a database file's structure which holds information required for multi-user lock detection.

## Syntax

```
CONVERT [TO <expN>]
```

## Usage

This command adds a character field called _dbaselock to the structure of the currently selected database file. The length of the field is determined by the numeric expression, which may be a number from 8 to 24. The default is 16.

The _dbaselock field reserves an area for the following values:

| | | |
|---|---|---|
| Count | = | A two-byte hexadecimal number used by the CHANGE() function. |
| Time | = | A three-byte hexadecimal number that records the time a lock was placed. |
| Date | = | A three-byte hexadecimal number that records the date a lock was placed. |
| Name | = | A zero- to 16-character representation of the log-in name of the computer that placed a lock, if a lock is active. |

The count, time, and date portions of the field always take the first eight characters.

If you CONVERT the _dbaselock field to the default of 16 characters, the log-in name will be eight characters long. If you CONVERT the field to the maximum of 24 characters, the log-in name will be 16 characters long. If you CONVERT the field to eight characters, no space is reserved for the log-in name, and the name is not written in the record.

Every time a record is updated, the count portion of _dbaselock is rewritten. If you use the CHANGE() function, the count portion of the field is read from the disk again and compared to the previous value, which was stored in memory when the record was initially read. If the values are different, another user has changed the record, and the CHANGE() function returns a logical true (.T.).

You can reset the value to false by repositioning the record pointer. GOTO RECNO() rereads the current record's _dbaselock field, and a subsequent CHANGE() command should return a false (.F.) unless another user has made another change in the interim.

The LKSYS() function returns the log-in name, date, and time portions of the _dbaselock field. It indicates who has locked the record or file, and when the lock was placed. If you place a file lock, the _dbaselock field of the first record in the database file contains the information used by CHANGE() and LKSYS().

### Tip

CONVERT copies the .dbf file to new file with a .cvt extension, then creates a new .dbf file containing the _dbaselock field. The .cvt file contains the original file structure before CONVERT.

### See Also

CHANGE(), FLOCK(), LKSYS(), LOCK(), NETWORK(), SET LOCK, SET REPROCESS, UNLOCK

# COPY

COPY duplicates all or part of an active database file, creating a new file. COPY is also the primary command used to export data to non-dBASE programs.

## Syntax

```
COPY TO <filename>
    [[TYPE] <file type>] [FIELDS <field list>]
    [<scope>] [FOR <condition>] [WHILE <condition>]
```

## Defaults

This command copies all records, including records marked for deletion, unless SET DELETED is ON, or unless you specify a scope, FOR, or WHILE clause to limit the records copied. All fields are copied, unless you specify a FIELDS list or use the SET FIELDS command. Memo fields are copied only if the new file is another dBASE IV database file.

A directory listing of new files created by the COPY command shows the dates on which the new files were copied.

If you do not enter a file type with the command, the file is copied to another dBASE IV database (.dbf) file.

## Usage

If the TO file is another dBASE IV database (.dbf) file, do *not* specify [TYPE]. This command automatically copies memo (.dbt) files when the TO file is a dBASE IV database file. If the TO file is an ASCII text file or a file supported by another software program, specify one of the file type options.

**NOTE**
*The COPY command does not verify that the files you build are compatible with another software program. You may specify field lengths, record lengths, number of fields, or number of records that are incompatible with other software. Note the file limitations of your other software program before exporting database files with COPY.*

## Options

The options for exported file types are:

- DELIMITED — Delimited Format ASCII file. Data is appended character by character starting on the left. Each record must end with a carriage return and line feed. A comma separates each field and, in addition, double quotation marks surround character data unless you specify another delimiter. This is the same as DELIMITED WITH ".
- DELIMITED [WITH <delimiter> ] — ASCII text (.txt) file with comma field separators and character fields enclosed within delimiter characters. All fields are separated by commas. Character fields are delimited with double quotes by default, unless you specify another delimiter character using the WITH <delimiter> option. Records in the text file are variable length, and every record ends with a carriage return and line feed.
- DELIMITED [WITH BLANK] — ASCII text (.txt) file with a single space character separating each field, but with no delimiters enclosing character fields. Records in the text file are variable length, and every record ends with a carriage return and line feed.
- SDF — System Data Format ASCII (.txt) file. Character data is not delimited, and fields are not separated with a character. Records are fixed length, the same length as the record in the database file, and every record ends with a carriage return and line feed.
- DBASEII — dBASE II database (.db2) file. The dBASE II file is given a .db2 file extension, rather than a .dbf extension, to distinguish it from the original dBASE IV file. You should rename the file to include a .dbf extension before you use it in dBASE II.
- DBMEMO3 — dBASE III PLUS format for database (.dbf) and memo field (.dbt) files. Once a database file and its memo file have been created or modified in dBASE IV, they cannot be opened in dBASE III PLUS. However, you can COPY them with TYPE DBMEMO3 and open the copies.
- RPD — RapidFile data (.rpd) file.
- FW2 — Framework II (.fw2) database.
- SYLK — MultiPlan spreadsheet formula. Database records are converted to MultiPlan rows, and database fields are converted to columns. No file extension is written with the output file.
- DIF — VisiCalc version 1 (.dif) file format. Database records are converted to VisiCalc rows, and database fields are converted to columns.

- WKS — Lotus 1-2-3 spreadsheet (.wks) format, release 1A. Database records are converted to Lotus 1-2-3 rows, and database fields are converted to columns.

When COPY is used to write any of the three supported spreadsheet formats (SYLK, DIF, WKS), the field names are written as column headers in the resulting file. The file is created in row major order.

## Special Cases

Use the EXPORT command, rather than the COPY command, to convert files to PFS:FILE. EXPORT and COPY both convert files to Framework II, dBASE II, and RapidFile file formats. COPY, however, cannot create a PFS:FILE form.

## Tips

Do not use the single letters A through J, or the letter M, as a database filename if you COPY TO a dBASE IV database file. These letters are reserved as default alias names. You can, however, specify AA (for example) as a database filename.

If a relation is active and you have specified fields from another work area with the SET FIELDS command or with the FIELDS clause, the resultant file contains the data from related records in other work areas.

## Example

To copy all the records in the Transact database file whose Client_id is C00002 to a database file called Temp:

```
. USE Transact
. COPY TO Temp FOR Client_id = 'C00002'
   2 records copied
. USE Temp.dbf
. LIST

Record# CLIENT_ID ORDER_ID DATE_TRANS INVOICED TOTAL_BILL
      1 C00002    87-107   02/12/87   .T.         1250.00
      2 C00002    87-110   03/09/87   .T.          175.00
```

## See Also

APPEND FROM, COPY FILE, COPY STRUCTURE, EXPORT, IMPORT, SET DELETED, SET FIELDS, SET SAFETY

# COPY FILE

COPY FILE creates a duplicate of any file.

## Syntax

COPY FILE <filename> TO <filename>

## Usage

You must specify the filenames and file extensions for both files. If you want to copy a file to another drive or directory, you must also provide the drive designator and directory name. You do not need to include the directory or drive designator of the first file, if it is already established with the SET PATH command or the PATH setting in the Config.db file.

You cannot use COPY FILE to copy an open file.

## Tips

If you copy a database file that has memo fields, you must copy the associated memo (.dbt) and production (.mdx) files separately. You may use the COPY command to copy records from an open database file.

Do not use the single letters A through J, or the letter M, as database filenames because they are reserved as default alias names. You can, however, specify AA (for example) as a database filename.

## Example

To make a duplicate of a dBASE program file:

```
. COPY FILE Accts.prg TO Oldaccts.prg
2123 bytes copied
```

## See Also

COPY

# COPY INDEXES

COPY INDEXES converts a list of index (.ndx) files into file tags in a single .mdx (multiple index) file.

## Syntax

COPY INDEXES <.ndx file list> [TO <.mdx filename>]

## Usage

If you do not specify an .mdx file with the TO clause, the tag is written to the production .mdx file. If a production .mdx file does not exist, it is created with the same name as the active database file, and the database file header is updated to indicate the presence of a production .mdx file.

If you use a TO clause, the tag is written to the specified .mdx file. If the .mdx file you supply in the TO clause does not exist, a new .mdx file is created and given the filename specified in the TO clause.

You may copy a maximum of 47 index (.ndx) files to tags in an .mdx file with one COPY INDEXES command, because each .mdx file may contain 47 tags.

## Special Cases

In a network environment, the database file must be in exclusive use before you attempt to copy an .ndx file to an .mdx tag.

## Example

To convert the Cus_name index file of the Client database file to a tag in the production .mdx file:

```
. USE Client INDEX Cus_name
. COPY INDEX Cus_name
100% indexed        8 records indexed
. DISPLAY STATUS
Currently Selected Database:
Select area: 1, Database in Use: C:\DBASE\CLIENT.DBF Alias: CLIENT
              Index file:    C:\DBASE\CUS_NAME.NDX  Key: Lastname+Firstname
Production     MDX file:    C:\DBASE\CLIENT.MDX
               Index TAG:      CLIENT  Key:  CLIENT
               Index TAG:      CLIENT_ID  Key:  CLIENT_ID
               Index TAG:      CUS_NAME  Key:  Lastname+Firstname
               Memo file:    C:\DBASE\CLIENT.DBT
```

### See Also

COPY TAG, INDEX, KEY(), MDX(), NDX(), SET INDEX, SET ORDER, TAG(), USE

# COPY MEMO

COPY MEMO copies the information from a single memo field to another file.

## Syntax

COPY MEMO <memo field name> TO <filename> [ADDITIVE]

## Default

If you do not provide an extension for the TO file, a .txt extension is written.

## Usage

This command exports the information from a memo field in the current record to a file on disk.

If you want to copy a file to another drive or directory, you must provide the drive designator and directory name.

You can precede the field name with an alias, and copy memo fields from other work areas. If the field is contained in a SET FIELDS list, you do not have to use the alias.

## Options

ADDITIVE causes the contents of the memo field to be appended to the end of the named file. If you do not use the ADDITIVE option and the filename already exists on disk, the contents of the memo field overwrite any existing information in the file.

This command writes to a file without warning, if SET SAFETY is OFF. If SET SAFETY is ON, and a file of the same name exists in the target directory, you are prompted with a warning message before the file is written.

## Example

To write the information contained in the field Clien_hist to a file named Cus_text, appending the information in the current record to information that already exists in the file:

```
. COPY MEMO Clien_hist TO Cus_text ADDITIVE
```

## See Also

APPEND MEMO, COPY, COPY FILE

# COPY STRUCTURE

COPY STRUCTURE copies the structure of the active database file to a new file, but does not copy any records.

## Syntax

COPY STRUCTURE TO <filename> [FIELDS <field list> ]

## Defaults

The filename must include the drive designator and directory, if you want the resultant file written to a drive or directory other than the default. Unless otherwise specified, the TO file is assigned a .dbf extension.

## Usage

This command copies the entire database file structure unless limited by the FIELDS option. The result is another database file with either an identical structure to the first file, or, if you specify the FIELDS option, with a subset of the fields in the first file. No records are copied.

If SET SAFETY is OFF, a new database file will overwrite an existing file without warning.

## Tip

You may COPY STRUCTURE under program control to create temporary database files. You may then add records to the transaction file with the APPEND command, or add blank records with APPEND BLANK and place data in the records with REPLACE.

## Example

To copy the structure of the Client database file to a file called Temp:

```
. USE Client
. COPY STRUCTURE TO Temp
```

## See Also

APPEND, APPEND BLANK, APPEND FROM, DISPLAY STRUCTURE, REPLACE, SET SAFETY

# COPY STRUCTURE EXTENDED

COPY STRUCTURE EXTENDED creates a new database file whose records contain the structure of the current file.

## Syntax

COPY TO <filename> STRUCTURE EXTENDED

## Usage

This command creates a database file with five fields: FIELD_NAME, FIELD_TYPE, FIELD_LEN, FIELD_DEC, and FIELD_IDX. Records in the new file contain the field name, data type, field width, number of decimal places (in a numeric field), and index flag for each field in the active database file.

This command is most often used within application programs in conjunction with the CREATE FROM command. CREATE FROM creates a database file from the extended structure file. You can thereby create a database file in a program without using the interactive CREATE or MODIFY STRUCTURE commands.

Because extended structure files created in dBASE IV contain a FIELD_IDX field, they are not compatible with dBASE III PLUS. The structures of database files created with earlier versions of dBASE did not include index flags.

## Example

See the CREATE FROM command.

## See Also

APPEND FROM, CREATE FROM, LIST/DISPLAY STRUCTURE

# COPY TAG

COPY TAG converts multiple index (.mdx) file tags into index (.ndx) files.

## Syntax

COPY TAG <tag name> [OF .mdx filename] TO <.ndx filename>

## Default

Using the OF clause, you may specify the .mdx file that contains the tag.

## Usage

The database file must be in use, because COPY TAG recreates the .ndx file from the expression contained in the .mdx file tag. The index tag being copied must come from an open .mdx file. Only one tag can be copied at a time.

COPY TAG respects the limit of 10 open .ndx files for the current .dbf file. Less than 10 .ndx files may be open when creating a new one with COPY TAG.

## Special Cases

In a network environment, the database file must be in exclusive use before you attempt to copy .mdx tags to .ndx files.

## Example

To copy the Order_id tag from the Stock .mdx file to an .ndx file called Items_id:

```
. USE Stock
. COPY TAG Order_id TO Items_id.ndx
  100% indexed        17 records indexed
. SET INDEX TO Items_id
. DISPLAY STATUS
Currently Selected Database:
Select area:   1, Database in Use: C:\SET\STOCK.DBF   Alias: STOCK
    Master Index file:  C:\SET\ITEMS_ID.NDX Key: ORDER_ID
Production   MDX file:  C:\SET\STOCK.MDX
            Index TAG:     ORDER_ID   Key: ORDER_ID
            Index TAG:     PART_NAME  Key: PART_NAME
```

## See Also

COPY INDEX, INDEX, MDX(), NDX(), SET INDEX, SET ORDER, TAG()

# COPY TO ARRAY

COPY TO ARRAY fills an existing array with the contents of one or more records from the active database file.

## Syntax

COPY TO ARRAY <array name> [FIELDS <fields list>]
    [<scope>] [FOR <condition>] [WHILE <condition>]

## Usage

This command copies selected records and fields from a database file into an existing array.

For each record in the database file, the first field is stored in the first column, the second field in the second column, and so on. Each record becomes a row in the array. This process continues until there are either no more fields or no more array columns. If the database file has more fields than the array can hold, the excess fields are not stored. If the database file has fewer fields than the array, the excess array elements remain unchanged. Memo fields cannot be copied to an array.

If you declare a single-dimensioned array, such as

```
. DECLARE Transact[5]
```

COPY TO ARRAY will only be able to copy the first five fields of one record to the array.

Unless you specify a scope, the process begins with the first record in the database file and continues until there are either no more records in the database file or there no more rows in the array.

The data types of the array elements will be the same as the corresponding field types in the database file.

## Options

This command attempts to copy all fields (except memo fields), unless you use the FIELDS option or the SET FIELDS command.

If you use the FOR clause, the condition is evaluated before each record in the database file is copied to the array. A record is copied to the array only if the condition evaluates to a logical true (.T.). If you use the WHILE clause, no further information is copied once the condition evaluates to a logical false (.F.).

## Example

To COPY the records in the Transact database file that have L0001 for the Client_id, first determine how many elements the array will require, define an array called Records, and then execute the COPY TO ARRAY command:

```
. USE Transact                  && Transact.dbf has five fields.
. COUNT FOR Client_id = "L00001" TO mcnt
    2 records
. DECLARE Records[mcnt,5]       && Define the array.
. COPY TO ARRAY Records FOR Client_id = "L00001"
    2 records copied
```

## See Also

APPEND FROM ARRAY, COPY FILE, COPY STRUCTURE, EXPORT, SET DELETED, SET FIELDS, SET SAFETY

# COUNT

COUNT tallies the number of records in the active database file that match specified conditions.

## Syntax

COUNT [TO <memvar>] [<scope>] [FOR <condition>]
    [WHILE <condition>]

## Usage

If SET TALK is ON, this command counts the number of records and displays the tally. If you specify a condition with a FOR or WHILE clause, or limit the number of records with a scope, the tally indicates the number of records that meet the condition or fall within the scope.

If you include the TO clause, COUNT creates a memory variable (if necessary) and stores the tally, as a type N (Binary Coded Decimal) number, to this memory variable.

## Special Cases

In a network environment, COUNT automatically locks the file during its operation if SET LOCK is ON (the default), and unlocks it after the count is complete. If SET LOCK is OFF, a COUNT can still be performed, but the result may not be reliable if another user is changing the database file.

## Example

To determine the number of March orders in the Transact database file:

```
. USE Transact
. COUNT FOR LIKE("03/??/87",DTOC(Date_trans)) TO March_cnt
    6 records
. ? "There were "+LTRIM(STR(March_cnt,3,0))+" orders in March."
There were 6 orders in March.
```

## See Also

AVERAGE, CALCULATE, RECCOUNT(), SUM

# CREATE or MODIFY STRUCTURE

CREATE or MODIFY STRUCTURE gives you access to the database file design screen.

CREATE allows you to build a structure for a new database file. MODIFY STRUCTURE allows you to modify the structure of a previously created database file. Both commands provide the same screen for designing the file structure.

The structure of a database file is the definition of field names, field types, field lengths, number of decimal places (for numeric fields), and a flag indicating the presence of an .mdx tag for each field.

## Syntax

CREATE <filename> /MODIFY STRUCTURE

## Defaults

Unless you explicitly provide a drive and directory with the filename, CREATE writes the new database file in the default drive and directory location. Unless you specify a different extension, the database file is given a .dbf extension.

You do not specify a filename with MODIFY STRUCTURE. This command can only modify the structure of the active database file.

If a catalog is active when you create a database file, the file will be added to the catalog.

## Usage

CREATE <newfile> and MODIFY STRUCTURE provide the same design screen for creating or changing the database file structure. The structure contains definitions for each field in the database file.

You define the structure of a new database file by providing the following information for each field:

Field Name
Type
Width
Decimal Places (for a numeric field)
Field Index Flag (if true, a tag is added to the production .mdx file,
    indexed on this field)

The field name may be up to 10 characters long, and may consist of letters, numbers, and the underscore character. The field name cannot contain embedded blank characters and the first character of the field name must be a letter. When you have finished entering the field name, press ↵.

You determine the field type by entering the first letter of the data type (**C**haracter, **N**umeric [Binary Coded Decimal], **F**loating point numeric, **L**ogical, **D**ate, or **M**emo), or by pressing the **Spacebar** until the desired data type appears and then selecting it by pressing ↵.

You must specify a field width for numeric and character fields. This is the maximum number of digits or characters you intend to enter in the field. Character fields may be up to 254 characters long; numeric fields may be up to 20 digits, including the sign and a decimal point.

Logical, Date, and Memo fields all have predefined widths. A Logical field is one byte wide. Date fields are always 8 bytes. Memo fields are automatically assigned a length of 10 bytes in the database file, although each memo field entry may contain up to 512K. Data you enter in memo fields is not stored in the database (.dbf) file, but in an associated memo (.dbt) text file. The 10 bytes identify the location of memo field entries in the memo file.

If you specify Y in the Index column of the design screen, dBASE IV will create an index tag on that field in the production .mdx file.

You can define a record having up to 255 fields. The maximum size of a record is 4,000 bytes, not including memo field entries. Each character position in a field takes up one byte.

Instructions and error messages appear at the bottom of the screen. The pull-down menus at the top of the screen allow you to work directly with the database file structure and records. You may print the database file structure, create indexes, sort the file, remove indexes, and append records to the file.

## Tips

MODIFY STRUCTURE makes a backup copy of the database file with a .bak extension, and a backup of the .dbt file with a .tbk extension. After the structure modifications are completed, this command appends the contents from the backup files into the modified database file. Since MODIFY STRUCTURE will not be able to create backup files if the disk or current directory is full, you should make sure that you have enough space available for the backup files before you modify a file's structure.

Because MODIFY STRUCTURE appends data into the new file, you may lose your data if you interrupt your computer while the command is saving changes.

## Special Cases

In a network environment, the database file must be in exclusive use before you can modify its structure.

You should not change a field's name and its width or type at the same time. If you do, dBASE IV will not be able to append data from the old field, and your new field will be blank. Change the name of a field, save the file, then use MODIFY STRUCTURE again to change the field's width or data type.

You also should not insert or delete fields from a database file and change field names at the same time. If you change field names, MODIFY STRUCTURE appends data from the old file by using the field position in the file. If you insert or delete fields as well as changing field names, therefore, you are changing field positions and could lose data. You can, however, change field widths or data types the same time as you insert or delete fields. In those cases, since MODIFY STRUCTURE appends data by field name, the data will be appended correctly.

dBASE IV will successfully convert data for a number of field type conversions. If you change field types, however, keep a backup copy of your original file, and check your new files to make sure the data has converted correctly.

If you convert numeric fields to character fields, dBASE IV will convert numbers from the numeric fields to character strings. If you convert a character field to a numeric field, dBASE IV will convert numeric characters in each record to digits until it encounters a non-numeric character. If the first character in a character field is a letter, the converted numeric field will contain zero.

You can convert logical fields to character fields, or vice versa. You cannot convert logical fields to numeric fields. You can also convert character strings which are formatted as a date (for example, mm/dd/yy or mm-dd-yy) to a date field, or convert date fields to character fields.

If you modify the field name, length, or type of any fields that have an associated tag in the production .mdx file, the tag is rebuilt.

In general, regarding the conversion of field data types, dBASE will attempt to make a conversion you request, but the conversion must be a sensible one or data may be lost. Numeric data can easily be handled as characters, but logical data, for example, cannot become numeric.

If you create a new database file or modify an existing file, you must use the SQL DBDEFINE command before using the file in SQL mode. The DBDEFINE command updates the system catalogs that SQL uses to access the file.

### See Also

APPEND FROM, APPEND MEMO, COPY STRUCTURE, COPY STRUCTURE EXTENDED, CREATE FROM, SET BLOCKSIZE, SET SAFETY, SET SQL

# CREATE FROM

CREATE FROM forms a new database file from the structure created with the COPY STRUCTURE EXTENDED command.

## Syntax

CREATE <filename> FROM <structure extended file>

## Usage

This command is most often used in application programs in conjunction with the COPY STRUCTURE EXTENDED command. CREATE FROM creates a database file from the extended structure file, without using the interactive CREATE or MODIFY STRUCTURE commands.

The file created with CREATE FROM becomes the active database file in the currently selected work area. If the CREATE FROM operation fails for any reason, no database file will be open in the current work area.

## Tip

Do not use the letters A through J, or the letter M, as database filenames, because they are reserved as default alias names. You can, however, use AA (for example) as a filename.

## Example

In this example, you use the COPY STRUCTURE EXTENDED command to create the Newnames database file from Client.dbf. Then you create a new database file with the same structure as Client.dbf, using CREATE FROM.

```
. USE Client
. COPY STRUCTURE EXTENDED TO Newnames
. USE Newnames
. CREATE Nclient FROM Newnames
. DISPLAY STRUCTURE
Structure for database: C:\DBASE\NCLIENT.DBF
Number of data records:   0
Date of last update   :   11/05/87
Field  Field Name  Type       Width  Dec  Index
    1  CLIENT_ID   Character      6           Y
    2  CLIENT      Character     30           Y
    3  LASTNAME    Character     15           N
    4  FIRSTNAME   Character     15           N
    5  ADDRESS     Character     30           N
    6  CITY        Character     20           N
    7  STATE       Character      2           N
    8  ZIP         Character     10           N
    9  PHONE       Character     13           N
   10  CLIEN_HIST  Memo          10           N
** Total **                     152
```

## See Also

COPY, COPY STRUCTURE, COPY STRUCTURE EXTENDED, LIST/DISPLAY STRUCTURE

# CREATE VIEW FROM ENVIRONMENT

CREATE VIEW FROM ENVIRONMENT builds a view (.vue) file that is compatible with dBASE III PLUS.

## Syntax

CREATE VIEW <.vue filename> FROM ENVIRONMENT

## Defaults

Unless otherwise specified, this command supplies a .vue extension to the file it creates. If a catalog is open, and SET CATALOG is ON, dBASE IV adds the view file to the catalog.

## Usage

The dBASE III PLUS CREATE VIEW FROM ENVIRONMENT command allowed you to create .vue files that saved information about the current selection of work areas: open database, format, and index files; relations; field lists; and filter conditions. The dBASE III PLUS SET VIEW command activated the environment saved in the .vue file by opening the files and re-establishing the field lists, relations, and filter conditions.

Although the dBASE IV query designer, which is accessible from the CREATE/MODIFY QUERY/VIEW command, provides much more capability than .vue files allowed, dBASE IV also allows you to create .vue files from the current environment for compatibility with dBASE III PLUS applications.

CREATE VIEW FROM ENVIRONMENT builds a view (.vue) file that saves the following information from the currently active environment:

- All open database files, index files, and the work area of each file
- All relations between the database files
- The currently selected work area number
- The active field list
- The open format (.fmt) file, if any
- Filter conditions in effect

You must open the files and establish the field lists, filter conditions, and relations before using CREATE VIEW FROM ENVIRONMENT.

SET VIEW TO a .vue file will activate the view that CREATE VIEW FROM ENVIRONMENT saved.

To deactivate the view file, open a different view file or type CLOSE DATABASES. This closes the open database files and their associated files that form the view file.

## See Also

SELECT, SET FIELDS, SET FORMAT, SET INDEX, SET RELATION, SET VIEW, USE

# CREATE/MODIFY APPLICATION

CREATE/MODIFY APPLICATION gives you access to the dBASE IV Applications Generator, which generates the code needed to tie objects, such as database files, index files, queries, reports, forms, menus, and lists, together in one application.

## Syntax

CREATE/MODIFY APPLICATION <filename> /?

## Defaults

You must provide a filename for the application, or use the ? (query clause), in the command line. If you provide an application filename and are creating a new application, dBASE IV uses this name in the **Application Definition** dialog box. You can, however, change the application name in the dialog box.

## Usage

The CREATE APPLICATION and MODIFY APPLICATION commands are identical. The presence of an application object (.app) file, rather than the command verb you use, determines whether a create or modify operation will occur. If the .app object file exists, this command allows you to modify it; if the .app object file does not exist, this command allows you to create a new one.

Besides the application object, you may create the following objects with the Applications Generator:

Table 2-4 Objects created with the Applications Generator

| Object | Extension | Description |
|---|---|---|
| Horizontal bar menu | .bar | Menu items that appear across the screen |
| Pop-up menu | .pop | Menu items that appear vertically in a frame |
| Files lists | .fil | A list of files from which you may choose |
| Structure list | .str | A list of fields in the current database file or view from which you may choose |
| Values list | .val | A list of values that a field may contain |
| Batch process | .bch | A series of actions associated with a menu item or list that your application may perform |

If a catalog is open, only the application (.app) object that you create is added to the catalog. Any other objects that you create (.bar, .pop, .fil, .str, .val, or .bch) are added to the current subdirectory.

## Options

If a catalog is open, the ? symbol queries the catalog for all available .app object files. If a catalog is not open, the ? symbol presents all .app object files on the disk. You can then choose the application you wish to modify.

## Special Cases

You may also enter the Applications Generator from the Control Center. If you enter the Applications Generator from CREATE/MODIFY APPLICATION, however, the exit options return control to the dot prompt or the next command in a program file, not to the Control Center.

## CREATE/MODIFY APPLICATION

If you select the **<create>** marker from the **Applications** panel in the Control Center, you can choose to create a new application with either the program editor or the Applications Generator. CREATE/MODIFY APPLICATION, however, brings you directly to the Applications Generator. To reach the program editor, you must type MODIFY COMMAND.

### See Also

*Using the dBASE IV Applications Generator* contains further information about what applications are and how to create them.

# CREATE/MODIFY LABEL

CREATE/MODIFY LABEL gives you access to the label designer. The label designer allows you to create label form (.lbl) files using the fields specified in the current database file or in other related database files.

## Syntax

CREATE/MODIFY LABEL <filename> /?

## Defaults

Unless you specify otherwise, dBASE IV creates the file with a .lbl extension and generates a file with an .lbg file extension. If a catalog is open and SET CATALOG is ON, dBASE IV adds the label file to the catalog.

## Usage

Use the CREATE LABEL command to create a new label form (.lbl) file. The .lbl file contains all the information needed to set up a display of the label design which can later be changed, and to print labels using data from a database file or view.

You specify the size of the label you want to create and the number of printed lines on each label. These values should match the label forms or paper on which you want to print.

The CREATE LABEL and MODIFY LABEL commands are identical. The presence of a label form file, rather than the command verb you use, determines whether a create or modify operation will occur. If the .lbl file exists, this command modifies it; if the .lbl file does not exist, this command creates one.

Using CREATE/MODIFY LABEL, you select and lay out the information you want on each label. When you save the label form, CREATE/MODIFY LABEL creates a file containing the dBASE IV code that prints labels. This file has the same name as the label form file, but with an .lbg file extension. When you first print labels with the LABEL FORM command, an .lbo file, which contains the compiled object code of the .lbg file, is written to disk. The LABEL FORM command subsequently uses this .lbo file whenever you print labels with this form.

## Options

If a catalog is open, the ? symbol queries the catalog for all available .lbl files associated with the active database file or view. If a catalog is not open, the ? symbol presents all .lbl files on the disk. You can then choose the label form file you wish to modify.

## Special Cases

You may also enter the label designer from the Control Center. If you enter the label designer from CREATE/MODIFY LABEL, however, the exit options return control to the dot prompt or the next command in a program file, not to the Control Center.

If you erase the label form (.lbl) file, you will not be able to use the CREATE or MODIFY LABEL commands to edit the file. An attempt to edit the file will generate a new label file if the old label file cannot be found.

Although you can use MODIFY COMMAND or a text editor to make changes directly to the generated label (.lbg) file, the changes will not be made to the .lbl or .lbo files. You use LABEL FORM to compile a new .lbo file. LABEL FORM compares the timestamps of .lbo and .lbg files to know when a new .lbo should be compiled.

If you modify a label form (.lbl) file that was created in dBASE III PLUS, the file is converted to dBASE IV format, and the original dBASE III PLUS .lbl file is saved with an .lb3 extension. You cannot run dBASE IV labels in dBASE III PLUS. Label forms created with dBASE III PLUS will run in dBASE IV without modification.

## See Also

LABEL FORM, SET CATALOG, SET SAFETY, SET VIEW

*Using the Menu System* contains further information on creating labels with the dBASE IV label designer.

# CREATE/MODIFY QUERY/VIEW

CREATE/MODIFY QUERY or CREATE/MODIFY VIEW gives you access to the query design screen, which allows you to create query (.qbe) files that extract records matching specified conditions, or to create update query (.upd) files that can modify records in the database file.

## Syntax

```
CREATE/MODIFY QUERY <filename> /?
    or
CREATE/MODIFY VIEW <filename> /?
```

## Defaults

Unless you specify otherwise, dBASE IV writes the query file with a .qbe or .upd extension. If a catalog is open and SET CATALOG is ON, the query file is added to the catalog.

## Usage

A .qbe file allows only the records that meet the specified conditions to be displayed when subsequent commands are issued. A .upd file contains instructions for updating records in the database file.

The new query file will have a .upd extension only if you placed an update operator under the name of the file in the design screen's file skeleton.

You may use either the CREATE/MODIFY QUERY or CREATE/MODIFY VIEW command to access the query design screen.

The CREATE and MODIFY forms of the command are identical. The presence of a query (either .qbe or .upd) file, rather than the command verb you use, determines whether a create or modify operation will occur. If the .qbe or .upd file exists, this command asks if you want to modify it; if the .qbe or .upd file does not exist, this command creates one.

dBASE IV views are a superset of the queries and views created in earlier versions of dBASE. The .vue files created by earlier versions can be read by this dBASE IV command, and new .qbe files will be created from them. The new .qbe files cannot, however, be read by earlier versions of dBASE.

If you do not specify an extension, the command first looks for a .qbe or .upd file previously created by the queries design screen. If a .qbe or .upd file cannot be found, this command looks for a .vue file created with an earlier version of dBASE. If a .vue file also cannot be found, this command creates a new file with either a .qbe or .upd extension.

When you first use SET VIEW to activate a .qbe file, a .qbo file is written to disk. The SET VIEW command subsequently uses the .qbo file whenever you activate this query.

When you use DO to perform the update query, a .dbo file is written to disk. You may rename this .dbo extension to .upo if you want to keep your update query files separate from your program files. (You must rename the extension to .upo, however, in order to add this file to the **Queries** panel of the Control Center.

## Options

If a catalog is open, the ? symbol queries the catalog for all available .qbe and .upd files that are associated with the active database file. If a catalog is not open, the ? symbol presents all .qbe and .upd files on the disk. You can then choose the query file you wish to modify.

## Special Cases

You may also enter the query designer from the Control Center. If you enter the query designer from CREATE/MODIFY QUERY, however, the exit options return control to the dot prompt or the next command in a program file, not to the Control Center.

If you erase the .qbe or .upd files, you will not be able to use the CREATE/MODIFY QUERY/VIEW command to edit these files. An attempt to edit these files will generate new query files if the old query files cannot be found.

Although you can use MODIFY COMMAND or a text editor to make changes directly to the .qbe or .upd files, the changes will not be made to the .qbo or .upo files. You must delete the old object files, and use SET VIEW or DO to compile new object files.

## See Also

SET CATALOG, SET FILTER, SET SAFETY, SET VIEW

*Using the Menu System* contains further information on creating queries with the query designer.

# CREATE/MODIFY REPORT

CREATE/MODIFY REPORT gives you access to the report designer, which allows you to create report form (.frm) files using the fields specified in the current database file or in other related database files.

## Syntax

CREATE/MODIFY REPORT <filename> /?

## Defaults

Unless you specify otherwise, dBASE IV creates the file with an .frm extension. CREATE REPORT will always generate a file with an .frg file extension; however, you cannot override this. If a catalog is open and SET CATALOG is ON, the report form file is added to the catalog.

## Usage

Use the CREATE/MODIFY REPORT command to create a new report form (.frm) file. The report form file contains all the information needed to set up a display of the report design which can later be changed, and to print reports using data from a database file or view.

The CREATE REPORT and MODIFY REPORT commands are identical. The presence of a report form (.frm) file, rather than the command verb you use, determines whether a create or modify operation will occur. If the .frm file exists, this command modifies it; if the .frm file does not exist, this command creates one.

Using CREATE or MODIFY REPORT, you may select the information you want the report to contain, place fields where you would like them to print, group information together, and calculate statistical information on numeric expressions. When you save the report form, CREATE/MODIFY REPORT creates a file containing the dBASE IV code that prints a report. This file has the same name as the report form file, but with an .frg file extension. When you print the report, as with the REPORT FORM command, an .fro file, which contains the compiled object code of the .frg file, is written to disk. The REPORT FORM command subsequently uses this .fro file whenever you print a report with this form.

## Options

If a catalog is open, the ? symbol queries the catalog for all available .frm files associated with the active database file or view. If a catalog is not open, the ? symbol presents all .frm files on the disk. You can then choose the report form file you wish to modify.

### Special Cases

You may also enter the report designer from the Control Center. If you enter the report designer from CREATE/MODIFY REPORT, however, the exit options return control to the dot prompt or the next command in a program file, not to the Control Center.

If you erase the report form (.frm) file, you will not be able to use the CREATE or MODIFY REPORT commands to edit the file. An attempt to edit the file will generate a new report file if the old report file cannot be found.

Although you can use MODIFY COMMAND or a text editor to make changes directly to the generated report form (.frg) file, the changes will not be made to the .frm or .fro files. You use REPORT FORM to compile a new .fro file.

If you modify a report form (.frm) file that was created in dBASE III PLUS, the file is converted to dBASE IV format, and the original dBASE III PLUS .frm file is saved with an .fr3 extension. You cannot run dBASE IV reports in dBASE III PLUS.

### See Also

REPORT FORM, SET CATALOG, SET DESIGN, SET SAFETY, SET VIEW

*Using the Menu System* contains further information on creating reports with the report designer.

# CREATE/MODIFY SCREEN

CREATE/MODIFY SCREEN gives you access to the forms design screen, which allows you to create custom screen forms. These screen forms determine the way fields and other data appear on the screen when you use a full-screen editing command, such as EDIT or APPEND.

## Syntax

CREATE/MODIFY SCREEN <filename> /?

## Defaults

dBASE IV writes the screen file with an .scr extension and generates a format file with an .fmt extension unless you specify otherwise. If a catalog is open and SET CATALOG is ON, the query and format files are added to the catalog.

## Usage

Use the CREATE SCREEN command to create a screen (.scr) file and new format (.fmt) file. The screen file contains the information in a form that you can later edit, and the format file contains the dBASE IV commands to display the data on the screen. When you design or modify a screen, you are working with the screen file.

The CREATE SCREEN and MODIFY SCREEN commands are identical. The presence of a screen (.scr) file, rather than the command verb you use, determines whether a create or modify operation will occur. If the .scr file exists, this command modifies it; if the .scr file does not exist, this command creates one and generates a new format file.

Using CREATE or MODIFY SCREEN, you can position fields on the screen; display calculated fields and memory variables; and include additional text, boxes, and lines. When you save the screen form, CREATE/MODIFY SCREEN creates a format file containing @ commands to display and allow editing of the data. The format file has the same name as the screen file, but with an .fmt file extension. When you first use the format file with the SET FORMAT command, an .fmo file, which contains the compiled object code of the .fmt file, is written to disk. dBASE IV subsequently uses this .fmo file whenever you use SET FORMAT with a full-screen editing command.

You can open or close the format file you created at any time with the SET FORMAT command.

### Options

If a catalog is open, the ? symbol queries the catalog for all available .scr files associated with the active database file or view. If a catalog is not open, the ? symbol presents all .scr files on the disk. You can then choose the screen file you wish to modify.

### Special Cases

Unlike earlier versions of dBASE, CREATE/MODIFY SCREEN does not make any changes to the physical structure of the database file.

If you erase the screen file, you will not be able to use the CREATE or MODIFY SCREEN commands to edit the file. An attempt to edit the file will generate a new screen file if the old screen file cannot be found.

Although you can use MODIFY COMMAND or a text editor to make changes directly to the format (.fmt) file, the changes will not be made to the .scr or .fmo files. You use SET FORMAT to compile a new .fmo file.

If you modify a screen (.scr) file that was created in dBASE III PLUS, the file is converted to dBASE IV format, and the original dBASE III PLUS .scr file is saved with an .sc3 extension. You cannot modify dBASE IV screen files in dBASE III PLUS.

The format file that CREATE/MODIFY SCREEN generates has literal strings embedded in double quote marks. Because of the double quote marks inside a format file, you should avoid using them in the surface design of a screen form.

### See Also

@, APPEND, BROWSE, EDIT, INSERT, SET CATALOG, SET FORMAT, SET SAFETY, SET VIEW

*Using the Menu System* contains further information on using the screen designer.

# DEACTIVATE MENU

The DEACTIVATE MENU command deactivates the active bar menu and erases it from the screen, while leaving it in memory. It has no effect when executed from the dot prompt. It is used in ON SELECTION statements or in procedures called by ON SELECTION statements.

## Syntax

DEACTIVATE MENU

## Usage

This command does not require a menu name; it deactivates the only active menu and erases it from the screen. The screen returns to displaying whatever is under the deactivated menu.

A deactivated menu is not released from memory; you can reactivate it at any time with the ACTIVATE MENU command.

DEACTIVATE MENU returns control to the program line immediately following the one that activated the menu. If an ON PAD is in effect, this command will also deactivate all descendent popups and menus.

## Example

This example defines a bar menu with two selections. It can display a directory listing or, alternately, deactivate itself:

```
. DEFINE MENU Test MESSAGE "Test"
. DEFINE PAD Dir OF Test PROMPT "Directory" AT 0,0
. DEFINE PAD Deac OF Test PROMPT "Deactivate" AT 0,15
. ON SELECTION PAD Dir OF Test DIR
. ON SELECTION PAD Deac OF Test DEACTIVATE MENU
. ACTIVATE MENU Test
```

## See Also

ACTIVATE MENU, CLEAR MENUS, DEFINE MENU, ON SELECTION, RELEASE MENUS

# DEACTIVATE POPUP

The DEACTIVATE POPUP command erases the active pop-up menu from the screen while leaving it intact in memory. Any text that was covered by the popup is displayed again.

## Syntax

DEACTIVATE POPUP

## Usage

DEACTIVATE POPUP has no effect when executed from the dot prompt, because when a popup is active, you can be only in the active popup. If you press the **Esc** key, you bypass this command, deactivate the popup, and return to the dot prompt. The ON SELECTION statement can call a procedure that uses the DEACTIVATE POPUP command.

## Example

This example shows a popup called Exit_pop, with two options. Either option calls a procedure that evaluates the user's selection. Notice that the popup is deactivated as one of the options of the procedure file.

```
. DEFINE POPUP Exit_pop FROM 3,38
. DEFINE BAR 1 OF Exit_pop PROMPT "QUIT"
. DEFINE BAR 2 OF Exit_pop PROMPT "Exit to dot prompt"
. ON SELECTION POPUP Exit_pop DO Exit
. ACTIVATE POPUP Exit_pop


PROCEDURE Exit
DO CASE
   CASE BAR() = 1
      QUIT
   CASE BAR() = 2
      DEACTIVATE POPUP
ENDCASE
```

## See Also

ACTIVATE POPUP, CLEAR POPUPS, DEFINE POPUP, ON SELECTION, RELEASE POPUPS

# DEACTIVATE WINDOW

The DEACTIVATE WINDOW command deactivates specified windows and removes them from the screen, without releasing them from memory.

## Syntax

DEACTIVATE WINDOW <window name list> /ALL

## Usage

This command deactivates windows in the window name list by erasing them from the screen. The windows are not released from memory, and you can bring them back to the screen with the ACTIVATE WINDOW command.

When you DEACTIVATE a window, any window that was previously ACTIVATEd becomes current again. DEACTIVATEing all windows with the ALL option restores full-screen mode.

## See Also

ACTIVATE SCREEN, ACTIVATE WINDOW, DEFINE WINDOW, MOVE WINDOW, RESTORE WINDOW, SAVE WINDOW

# DEBUG

DEBUG gives you access to the dBASE IV program debugger.

## Syntax

DEBUG <filename>/<procedure name> [WITH <parameter list>]

## Usage

This command, like DO, executes a program or procedure, but also calls dBASE IV's full-screen debugger.

The debugger screen contains four windows that allow you to run a program or procedure and see the commands as they are executing, edit the program or procedure, set breakpoints to halt program execution, and display the results of expressions while the program is executing.

The screen is divided into a debug window, an edit window, a breakpoint window, and a display window.

The debug window, at the bottom of the screen, displays the current work area, database file, program file, procedure, record number, line number, master index file, and the **ACTION:** prompt. Pressing **Esc** or **Ctrl-End** anywhere on the screen returns you to the **ACTION:** prompt in the debug window.

If you enter an E at the **ACTION:** prompt, the edit window becomes active. This window, at the top of the screen, shows the program or procedure being executed. When the edit window is active, you can access the dBASE IV editor and make changes to the program. You must save the file to avoid losing the changes. Once the changes are made, the debugger continues execution from the old file. The debugger may execute a command line that is not the line you expect to be executed, depending on how you changed the file.

If you enter a B at the **ACTION:** prompt, the breakpoint window, on the right side of the screen, becomes active. You may enter one or more conditions in the breakpoint window that will be evaluated after each line of code is executed. If one of the conditions evaluates to true (.T.), the program is halted and the debug screen reappears.

If you enter a D at the **ACTION:** prompt, the display window, on the left side of the screen, becomes active. You may enter dBASE expressions on the left of this window. The results are displayed on the right of the display window.

At the **ACTION:** prompt, you may also enter the following:

- L Line — Specify which line to execute next.
- N Next — Execute the next command in the current procedure, then return to the **ACTION:** prompt. If there is a DO in the current procedure, the called procedure will execute outside the debugger environment, although the breakpoint watch will remain in effect. If you precede the N with a number, you direct the debugger to execute that number of commands in the current procedure before returning to the **ACTION:** prompt.
- P Program Trace — Show the program trace information, which includes the current program, procedure, and line number.
- Q Quit — Quit the debugger and cancel the program.
- R Run — Run the program until a breakpoint or error is encountered.
- S Step — Execute the next command, then return to the **ACTION:** prompt. If you precede the S with a number, you direct the debugger to step through that number of commands before returning to the **ACTION:** prompt. Unlike N, procedures called from the current procedure are executed within the debugger environment.
- X Exit — Exit the program and return to the dot prompt. From the dot prompt, RESUME passes control back to the debugger.

  You can only operate one DEBUG session at a time; it isn't possible to start a second while the first is suspended.
- ↵ — Execute either 1S or 1N. If a Step, S, was last executed, ↵ will execute a 1S. If a Next, N, was last executed, ↵ will execute a 1N. By default, ↵ executes a 1S.

**F1 Help** and **F9 Zoom** are toggle keys:

Pressing **F1 Help** anywhere on the screen brings up or removes the Help panel, a brief description of the debug commands you can enter at the **ACTION:** prompt.

Pressing **F9 Zoom** removes the debugger windows from the screen to show the underlying screen information, or replaces the debugger windows on the screen.

## Options

The WITH parameter allows you to pass parameters in the same way as DO. The parameter list may contain any valid dBASE IV expressions. Note that DEBUG with <parameter list> can accept a maximum of eight constants, which includes a filename. The number of variables is not limited.

### See Also

COMPILE, DO, MODIFY COMMAND, SET DEBUG, SET ECHO, SET PROCEDURE, SET STEP, SET TALK, SET TRAP

# DECLARE

DECLARE creates one- or two-dimensional arrays of memory variables.

## Syntax

DECLARE <array name 1> [{<number of rows>,}
{<number of columns>}] {,<array name 2> [{<number of rows>,}
{<number of columns>}] ...}

In this paradigm, the curly braces indicate optional items. The square brackets are a required part of the DECLARE command syntax.

## Defaults

The array is public if the DECLARE command is entered at the dot prompt, private if the command is in a program file. You may create a public array in a program file with the PUBLIC command.

```
PUBLIC ARRAY Parts[6,2]
```

creates a public array, called Parts, if the command is used in a program file.

```
DECLARE Parts[6,2]
```

creates a private array, called Parts, if you use the command in a program file. If you use the command from the dot prompt, the array is public.

## Usage

The array definition list consists of the array names and array dimensions. Like memory variables, array names can be up to ten characters long. They can contain letters, numbers, and underscores. They must begin with a letter and cannot contain embedded blank spaces. The array name cannot be the same as a dBASE IV command.

The array dimensions consist of one or two numbers in square brackets. The first number is the number of rows in the array; the second is the number of columns in the array. If only one number is used, the array is one-dimensional. If two are used, they are separated by a comma and the array is two-dimensional. The maximum number of dimensions is two.

```
. DECLARE Cost[15]
```

creates a one-dimensional array (a row).

Specifying just the number of columns in a row is the same as declaring a one-row array, as in:

```
DECLARE Cost[1,15]
```

Cost is the array name, and it contains fifteen elements. The elements are numbered starting at 1.

```
. DECLARE Items[8,3]
```

creates a two-dimensional array called Items, which has eight rows and three columns. It contains 24 elements, numbered by row and column position.

The DECLARE command creates a set of memory variables, each of which initially contains a logical false (.F.) value. Array elements assume a data type only when information is STOREd to them. For example:

```
. STORE {6/15/88} TO Mdate[2,2]
```

initializes the element Mdate[2,2] with a date value.

One array may contain elements of different data types.

**NOTE**
*Any array, no matter how complex, is still handled as a memory variable. This means you can replace an array with a variable of the same name without seeing a warning message or needing to approve the replacement. Use caution when you create or replace memory variables with names similar to those of arrays present in memory.*

The array name uses one slot from the same memvar pool used by other memory variables. Each array element, however, does not use a slot from this memvar pool, but is stored in a separate block allocated to hold the elements . If an array declaration exceeds available memory, the error message **Insufficient Memory** appears. After declaring an array, the elements are treated like any other memory variable. The elements are referred to by their array name and position in the array, beginning from left to right and top to bottom. For example, Cost[4] or Items[5,2] are sample element names.

All commands and functions which can be used with memory variables can also be used with array elements, as long as the array has been declared and the element is within the range of the array declaration. Some commands, such as COPY and APPEND, have special forms (COPY TO ARRAY and APPEND FROM ARRAY) to handle arrays. Commands that manipulate memory variables (such as CLEAR ALL, CLEAR MEMORY, LIST/DISPLAY MEMORY, RELEASE, RESTORE, and SAVE) support arrays as well as memory variables.

If you reference array coordinates that do not exist, the error message **Bad array dimension(s)** appears. If you reference an array name that has not been DECLAREd, the message **Not an array** appears.

## Examples

Using the Transact database file, store the number of orders and the sum of the Total_bill field to an array called Details:

```
. SET TALK OFF
. USE Transact
. DECLARE Details[2]
. CALCULATE CNT(), SUM(Total_bill) TO ARRAY Details
. ? LTRIM(STR(Details[1],8,0)), "orders for $" + LTRIM(STR(Details[2],9,2))
12 orders for $10080.00
```

Using the same database file in a program file, use two arrays to detail the breakdown of orders by Client_id:

```
USE Transact ORDER Client_id
DECLARE Orders[12,3]              && Declare one row for each client.
DECLARE Details[2]
Mcnt = 1
DO WHILE .NOT. EOF()
   Orders[Mcnt,1] = Client_id     && Save Client_id.
   CALCULATE CNT(), SUM(Total_bill) TO ARRAY Details;
      WHILE Client_id = Orders[Mcnt,1]
   Orders[Mcnt,2] = Details[1]    && Save count of orders.
   Orders[Mcnt,3] = Details[2]    && Save total for Client_id.
   Mcnt = Mcnt + 1
ENDDO
? "Client ID", "Orders" AT 12, "Total" AT 24 && Column headings.
Mclients = Mcnt - 1
Mcnt = 1
DO WHILE Mcnt <= Mclients
   ? Orders[Mcnt,1], STR(Orders[Mcnt,2],8,0) AT 10,;
      STR(Orders[Mcnt,3],9,2) AT 20
   Mcnt = Mcnt + 1
ENDDO
```

### See Also

APPEND, AVERAGE, CALCULATE, CLEAR ALL, CLEAR MEMORY, COPY, COUNT, LIST/DISPLAY MEMORY, PUBLIC, RELEASE, RESTORE, SAVE, SUM

# DEFINE BAR

The DEFINE BAR command defines a single option in a pop-up menu.

## Syntax

```
DEFINE BAR <line number> OF <popup name> PROMPT <expC>
    [MESSAGE <expC> ] [SKIP [FOR <condition> ]]
```

## Usage

A bar is a single prompt or option that appears in a defined pop-up. To use DEFINE BAR, you must not use the PROMPT FIELD, PROMPT FILES, or PROMPT STRUCTURE options of the DEFINE POPUP command. This is because the PROMPT FIELD options take the place of BARs and fill a defined pop-up window.

Use only positive whole numbers for the line numbers; fractional line numbers are truncated.

If you define a second bar prompt for a line number that already has a bar prompt, the new bar prompt overwrites the earlier one.

If you define a bar for a number that exceeds the total number of lines found in the pop-up window, then the prompts scroll vertically inside the pop-up window.

If a BAR value is missing, that row in the popup is left blank, and the selection bar skips over it.

If the length of the bar prompt exceeds the horizontal line length in the pop-up window, the prompt is truncated. Horizontal scrolling is not permitted in a pop-up window.

You must define at least one bar for a pop-up window; otherwise, the pop-up window is empty and cannot be activated.

The MESSAGE expression is displayed centered on the last line of the screen outside the pop-up window. The DEFINE BAR message overwrites any message text you have written with the DEFINE POPUP command. The message is limited to 79 characters; all excess characters are truncated. The message is tied to the bar prompt with which it is defined; it appears at the bottom of the screen when the cursor in the pop-up window is on the bar prompt that was defined in the same DEFINE BAR command.

Each bar prompt can have its own message line of 79 characters or less.

Use the SKIP option to display the desired BAR, but not allow its selection. Use the SKIP FOR option to display the BAR and allow its selection only when the FOR condition is true.

## Example

The following lines define menu choices for the View_pop pop-up menu:

```
. Medit = .F.
. DEFINE POPUP View_pop from 3,4 TO 8,19
. DEFINE BAR 1 OF View_pop PROMPT "Add new record"
. DEFINE BAR 2 OF View_pop PROMPT "Edit"
. DEFINE BAR 3 OF View_pop PROMPT REPLICATE("-",16) SKIP
. DEFINE BAR 4 OF View_pop PROMPT "Delete" SKIP FOR Medit
. ACTIVATE POPUP View_pop
```

BAR 3 displays a horizontal line to separate the Delete choice from the Add and Edit choices. The SKIP option is included to avoid selecting the separator line. A logical memory variable called Medit can be defined to make the Delete option display only while Medit evaluates to a logical true (.T.). The selection bar cannot be placed on bar 4 while Medit remains true, but can be selected when Medit evaluates to a logical false (.F.).

## See Also

ACTIVATE POPUP, BAR(), DEACTIVATE POPUP, DEFINE POPUP, ON SELECTION POPUP, POPUP(), PROMPT(), SHOW POPUP

# DEFINE BOX

The DEFINE BOX command defines a box to be printed around lines of text.

## Syntax

```
DEFINE BOX FROM <print column> TO <print column>
    HEIGHT <expN> [AT LINE <print line>] [SINGLE/DOUBLE
    /<border definition string>]
```

## Usage

Use this command to define a box around report text for enhanced appearance. This command is valid only for printed output.

This command defines the beginning column on the left and last column on the right, the beginning line for the top of the box, and the height for the box. If the beginning coordinates are less than the current coordinates, they are ignored and the box begins at the current line. Also, if no AT LINE is specified, the box begins at the current line.

The border definition string follows the same rules as the SET BORDER command. You may specify a list of character codes for the border string, as described in SET BORDER. The default border is a single line. The PANEL selection of SET BORDER is not supported.

To enable the printing of boxes, the _box system memory variable must be set to true (.T.).

## Example

See the SET BORDER command.

## See Also

SET BORDER

Chapter 1, "Essentials," describes how to control printer output with system memory variables.

Use DEFINE MENU in conjunction with the DEFINE PAD command to define a menu.

# DEFINE MENU

## Syntax

DEFINE MENU <menu name> [MESSAGE <expC>]

## Usage

This command is the first step in creating a bar menu; by itself it does not create a bar menu. This command can only assign a name to a bar menu and associate an optional message with the menu name.

Use this bar menu name with the DEFINE PAD command to define the menu pads and their messages.

The optional MESSAGE text field appears centered at the bottom of the screen. The message is limited to 79 characters; excess characters are truncated.

Each menu pad may have its own message, or one message may be used with all bar menu options. If a PAD is assigned a message, the message specified with the DEFINE MENU command is overwritten.

## Example

```
. DEFINE MENU Main
```

## See Also

DEFINE PAD, MENU(), ON PAD, ON SELECTION PAD, PAD()

# DEFINE PAD

Use the DEFINE PAD command to define a single pad in a bar menu. To define more than one pad in a menu, repeat the command with the same menu name until all the pads are defined.

## Syntax

```
DEFINE PAD <pad name> OF <menu name> PROMPT <expC>
    [AT <row>,<col>] [MESSAGE <expC>]
```

## Usage

Use this command to define each pad for a given bar menu. The pad name follows the naming rules for field and alias names. If you use an existing pad name to define a different pad, the earlier pad is overwritten.

The <menu name> option must be previously defined with the DEFINE MENU command.

The PROMPT text is displayed inside the menu option. The program adds a blank space to each side of the prompt message before displaying it.

Menu prompts can be positioned anywhere on the screen. The optional screen coordinates define the beginning point for the prompt text. You can create a vertical menu bar by using one set of column coordinates, but incrementing the row coordinates for each pad.

If you do not specify coordinates, the program places the first prompt at the upper left corner of the screen. It places each subsequent prompt on the same line, one space after the end of the previous prompt. SET SCOREBOARD OFF to prevent the SCOREBOARD information from writing over menu PADS on line one of the display.

To navigate the prompts, use the → and ← keys.

The MESSAGE option defines a message and associates it with the PAD. The message line can be up to 79 characters long; any excess characters are truncated. The message appears centered at the bottom of the screen when the cursor is on the pad associated with it. The message text overrides any other message defined with the DEFINE MENU command.

The total number of pads you can define is limited only by the available memory.

## Example

Before you define a pad, you define its menu. After defining pads, you define the pop-up menus for each one, then the bars that will appear on the pop-ups. The following program example creates a menu named Main and its four pads, with pop-ups for each pad and a bar for the first pop-up.

```
* Create a menu called Main and define its pads

DEFINE MENU Main
DEFINE PAD View OF Main PROMPT "Add/Edit" AT 2,4
DEFINE PAD Goto OF Main PROMPT "Goto/Search" AT 2,16
DEFINE PAD Print OF Main PROMPT "Print" AT 2,30
DEFINE PAD Exit OF Main PROMPT "Exit" AT 2,38

* Assign popups to the pads

ON PAD View OF Main ACTIVATE POPUP View_pop
ON PAD Goto OF Main ACTIVATE POPUP Goto_pop
ON PAD Print OF Main ACTIVATE POPUP Prin_pop
ON PAD Exit OF Main ACTIVATE POPUP Exit_pop

* Define the View pad's first popup and first bar

DEFINE POPUP View_pop FROM 3,4 MESSAGE "Select a View"
DEFINE BAR 1 OF View_pop PROMPT "Choose View"

* Take a look at the menu to see how it appears

ACTIVATE MENU Main
```

After this code executes, the ACTIVATE MENU command makes the menu appear on the screen.

## See Also

DEFINE MENU, ON PAD, ON SELECTION PAD, PAD (), SET SCOREBOARD

# DEFINE POPUP

A pop-up menu is a screen window containing special fields, messages and a border. The DEFINE POPUP command defines a pop-up window's name, location, border, prompts, and message line.

## Syntax

```
DEFINE POPUP <popup name> FROM <row1>,<col1>
    [TO <row2>,<col2>] [PROMPT FIELD <field name>
    /PROMPT FILES [LIKE <skeleton>]/PROMPT STRUCTURE]
    [MESSAGE <expC>]
```

## Usage

The DEFINE POPUP command arguments are as follows:

Pop-up menu names follow the same naming rules as the alias and field names. You must assign a name to the pop-up menu so that you can call the pop-up menu to the screen after you define it.

The FROM and TO coordinates define the top left and the bottom right corners of the pop-up window. This window covers up any other text that is displayed on the screen, including the status line. Because only one popup can be active, several popups may be defined at the same screen coordinates. Deactivated popups are erased from the screen.

The TO coordinates are optional; if you omit them, dBASE IV defines the window to be wide enough to accommodate the longest field and long enough to include the maximum number of lines. The screen limits a pop-up window to the last column (79) and the line above the status bar (22). If the status bar is off, the last line is the screen size, minus 1 (24 lines on a 25-line screen). The minimum size for a pop-up is one displayable row and one displayable column.

If you use the TO coordinates, prompts that are too long to fit in the pop-up window are truncated to fit. If all the prompts will not fit in the pop-up window, they scroll vertically within the pop-up menu window as you move the cursor.

There are three forms of the PROMPT option: PROMPT FIELD, PROMPT FILES, and PROMPT STRUCTURE. If you use any of the three when you define a pop-up menu, then you cannot later use the DEFINE BAR command with that pop-up menu name.

The PROMPT FIELD option displays the contents of the named field for each record of the database file in the pop-up window. You cannot use a memo field in the PROMPT FIELD option. You may, however, precede a field name with an alias.

If you enter the FILES option, the CATALOG filenames are displayed in the pop-up window. Specifying the LIKE < skeleton > parameter restricts the files displayed in the pop-up window to those that match the skeleton. Without the LIKE < skeleton > filter, all files in the active catalog are displayed.

If you use the STRUCTURE option, you see the defined fields in the pop-up window. Defined fields consist of all the fields in the active database file, or the fields in the SET FIELDS list if the SET FIELDS command is ON.

The MESSAGE expression is displayed centered in the bottom line of the screen, outside the pop-up window. The maximum message is 79 characters long; any excess characters are truncated. The message line is the only way to include a message text for the FIELD, FILES, or STRUCTURE options. This message has priority over any other text displayed by the SET MESSAGE TO command.

## Example

Here are four pop-up menus:

```
. DEFINE POPUP View_pop FROM 3,4 TO 8,19
. DEFINE POPUP Goto_pop FROM 3,16 TO 6,28
. DEFINE POPUP Prin_pop FROM 3,30 TO 7,42
. DEFINE POPUP Exit_pop FROM 3,38 TO 6,57
```

These pop-up menus have coordinates that would position them directly below corresponding pads on the menu bar .

## See Also

ACTIVATE POPUP, BAR(), DEFINE BAR, ON SELECTION POPUP, POPUP(), PROMPT()

# DEFINE WINDOW

The DEFINE WINDOW command defines windows, borders, and screen colors for windows.

## Syntax

DEFINE WINDOW <window name> FROM <row1>,<col1>
    TO <row2>,<col2> [DOUBLE/PANEL/NONE/
    <border definition string>] [COLOR [<standard>] [,<enhanced>]
    [,<frame>]]

## Usage

Use this command to define the screen coordinates and the display attributes of a window and its borders. The FROM coordinates determine the upper left row and column for the window, and the TO coordinates determine the bottom right row and column. Each time you activate a window, its coordinates are checked against the number of lines the screen can display, and the presence of the status line.

The border default is a single-line box. Alternately, you can define a double-line box, define an inverse video panel, or suppress borders altogether. Border definition strings use ASCII codes, as explained for the SET BORDER command.

If you change the SET BORDER parameters, the window retains the border format that was in effect when it was defined. If no color is specified, screen attributes follow the colors that were in effect on the screen when the window was defined.

Avoid using ASCII codes 7, 8, 10, 12, 13, 27, and 127 in border definitions. These codes display on the screen, but cause problems with print drivers, or if you use the **Shift-PrtSc** key to print out the screen contents.

The COLOR option allows you to set the foreground and background colors that will appear for standard and enhanced characters. The <frame> parameter, like other color settings, lets you set standard and enhanced attributes, but these apply to the window's frame or border.

You may store up to 20 window definitions in memory at one time.

## Example

This example opens a small window at the upper right corner of the screen. The sample border definition string uses the asterisk character (ASCII 42) for the borders, and the number 1 (ASCII 49) at the upper left corner as an optional window number. It uses a plus sign (ASCII 43) for the other three.

```
. DEFINE WINDOW W1 FROM 1,50 TO 10,79
     CHR(42),CHR(42),CHR(42),CHR(42),CHR(49),CHR(43),CHR(43),CHR(43)
. ACTIVATE WINDOW W1
```

### See Also

ACTIVATE WINDOW, DEACTIVATE WINDOW, RESTORE WINDOW, SAVE WINDOW, SET BORDER TO, SET COLOR

# DELETE

DELETE marks records in the active database file for deletion.

## Syntax

DELETE [ <scope> ] [FOR <condition> ] [WHILE <condition> ]

## Defaults

Unless otherwise specified by the scope or a FOR or WHILE clause, only the current record is marked for deletion.

## Usage

This command does not remove records from the database file. DELETE marks records for deletion, and PACK removes them from the database file. You can unmark records already marked for deletion with RECALL.

When you use DISPLAY and LIST to call up records, those marked for deletion are indicated by an asterisk (*) in the first position of the record.

With full-screen commands such as BROWSE or EDIT, records marked for deletion are indicated by **Del** on the status bar. In this mode, **Ctrl-U** both deletes and reinstates records.

## Record Pointer

DELETE does not reposition the record pointer, unless you give the scope, a FOR clause, or a WHILE clause. Therefore, if you're at the end of the file, as after LIST or DISPLAY ALL, issuing DELETE has no effect.

## Example

To mark record 6 for deletion:

```
. DELETE RECORD 6
   1 record deleted
. RECALL ALL
   1 record recalled
```

## See Also

DELETED(), PACK, RECALL, SET DELETED, ZAP

# DELETE TAG

DELETE TAG deletes the indicated tags from a multiple index (.mdx) file if tag names are specified, and closes index (.ndx) files if index file names are specified.

## Syntax

```
DELETE TAG <tag name 1> [OF <.mdx filename>]/<.ndx filename 1>
    [,<tag name 2> [OF <.mdx filename>]/<.ndx filename 2> ...]
```

## Usage

Multiple index (.mdx) files may contain up to 47 tags, each of which imposes an index order on the database file.

A production .mdx file is an .mdx file that is opened whenever the database file is USEd. Production .mdx files have the same name as their associated database files, but with an .mdx file extension. The database file header contains an indication that there is an associated production .mdx file.

If you no longer need one or more of the tag indexes in any active multiple index file, you can remove it with the DELETE TAG command. Deleting a tag index permanently removes it from the multiple index file, and restores space to the file. DELETE TAG opens a slot for a new tag index to be created in the .mdx file.

The multiple index file containing the tag must be open for DELETE TAG to work, but the tag that is being deleted does not have to be the controlling index.

If you delete all tags in a multiple index file, the .mdx file is also deleted. If you delete the production .mdx file by removing all its tags, the database file header is updated to indicate that no production .mdx file is associated with this database file. If a catalog is open, it is updated to reflect the changes.

Using the OF clause, you may specify the .mdx file that contains the tag. If the tag is contained in an .mdx file other than the production .mdx file, or if two open .mdx files contain the same tag name, you should include the OF clause in the command to indicate the correct tag for deletion. If a tag cannot be found, the error message **TAG not found** appears.

DELETE TAG operates differently when given index (.ndx) filenames. The index file is closed, not deleted from the disk. The result is similar to SET INDEX, although the operation is slightly different. The DELETE TAG syntax provides a convenient way to close certain .ndx files, when several are open, without closing and reopening the others. SET INDEX, however, closes all .ndx files and then reopens any files that should not be closed.

## Special Cases

In a network environment, the database file must be in exclusive use before you can issue DELETE TAG. If the database file is not in exclusive use, the error message **Exclusive use of database is required** appears.

## Example

To delete the Client tag from the Client production .mdx file:

```
. DELETE TAG Client OF Client
```

## See Also

COPY INDEX, COPY TAG, INDEX, MDX(), NDX(), ORDER(), SET INDEX, SET ORDER, TAG(), USE

# DIR

DIR displays directory information similar to that displayed by the DOS DIR command.

## Syntax

DIRECTORY/DIR [[ON] <drive>:] [[LIKE] [<path>] <skeleton>]

## Defaults

If you do not specify a filename or skeleton, the DIR command provides directory information only on database files. If you do not specify a path or drive, the DIR command provides information only on the current drive and directory.

## Usage

For database files, DIR displays the files, number of records, date of last update, file size (in bytes), total number of files displayed, total number of bytes for the displayed files, and the total number of bytes remaining on disk. If you specify any file type other than .dbf in the command, DIR displays only the filenames.

The directory may not reflect changes to newly changed files until after the file is closed, unless SET AUTOSAVE is ON.

## Examples

To display the database files in the current directory:

```
. DIR
```

To display all filenames in the current directory:

```
. DIR *.*
```

To display all compiled program files in the \SALES subdirectory:

```
. DIR \SALES\*.dbo
```

To display filenames that are three to five characters long where D is the third character:

```
. DIR ??D??.*
```

To display all .dbf files in another directory called Sales:

```
. DIR \SALES\
```

## See Also

LIST/DISPLAY FILES, SET AUTOSAVE

# DISPLAY Commands

The following DISPLAY commands are similar to their LIST counterparts. If you use the DISPLAY form of the command, only one screen of information is presented at a time, and a prompt appears asking you to press a key to see the next screen.

Both the DISPLAY and LIST forms allow you to send output to a disk file by using the TO FILE <filename> option.

Please refer to the following LIST commands for a discussion of both the LIST and DISPLAY forms:

LIST/DISPLAY

LIST/DISPLAY FILES

LIST/DISPLAY HISTORY

LIST/DISPLAY MEMORY

LIST/DISPLAY STATUS

LIST/DISPLAY STRUCTURE

LIST/DISPLAY USERS

# DO

DO executes a dBASE command file or procedure. If the command file or procedure file has not been COMPILEd, it is first parsed, then compiled and saved as an object file with a .dbo extension; then, the .dbo file is executed.

DO may also pass parameters to the named program.

## Syntax

```
DO <program filename>/<procedure name>
    [WITH <parameter list>]
```

## Defaults

Unless the file is on the default drive or a path is set, the filename must include the drive designator. The filename must also include a path, if the file is not on the default directory or on a path set with the SET PATH command.

You should not, however, precede the names of procedures within a procedure file with a drive designator or path. The drive designator and path for procedure files should be stated in the SET PROCEDURE command.

The following search order is used when you issue a DO command.

1. DO searches for a procedure in the current .dbo file, if one is being executed.
2. DO searches for a procedure in a procedure file, if a SET PROCEDURE command activated one.
3. DO searches for a procedure in other open .dbo files.
4. DO searches for a .dbo file with the associated name.
5. DO searches for a .prg file with the associated name, compiles it, and then executes the resultant .dbo file.
6. DO searches for a .prs file with the associated name, compiles it, and then executes the resulting .dbo file.

DO determines whether the file is an object or source file, and executes the file if it is an object (.dbo) file, or compiles and executes the file if it is a source (.prg or .prs) file.

## Usage

dBASE IV supports the concept of procedures within any program file by maintaining a procedure list in every object (.dbo) file. If a source file starts with a command other than PROCEDURE or FUNCTION, the code is compiled as a procedure and added to the procedure list in the object file with the same name as the source file. A typical .prg file such as:

```
* Main.prg
? "MAIN"
RETURN
```

is compiled into a .dbo file containing one procedure, Main. When you enter DO MAIN, this name is used to locate the .dbo file, and then used to locate the procedure within the .dbo file.

A source file can include more than one procedure, such as:

```
* Main.prg
? "MAIN"
DO Subb
RETURN

PROCEDURE Subb
? "SUBB"
RETURN
```

Note that only commands found at the beginning of a file are given the default procedure name. Commands between RETURN and PROCEDURE cause a compile-time warning. You may want to keep this code in the program file, but DO will not execute these commands.

Any procedure found in an active .dbo file is available to the DO command. If A.dbo calls B.dbo calls C.dbo, all the procedures defined within A, B, and C are available to any procedure in C. dBASE IV still supports SET PROCEDURE TO from dBASE III and dBASE III PLUS, although this command is only required to gain access to procedures in a file not activated by DO <filename>.

You may have a maximum of 32 active .dbo files. A .dbo file is active if you open it with SET PROCEDURE TO, or if a RETURN can pass control back to it.

The total number of open files, including database files, index files, format files, and command files, is determined by the FILES setting in the Config.sys file. In a DOS environment five files are reserved; therefore, you can have up to 94 files open if you set FILES = 99, which is the maximum setting in Config.sys.

When the program called by DO is complete, control returns to the calling program (or to the dot prompt or Control Center if the DO command was issued from either of those points).

Memory variables and arrays created in the called program must be declared PUBLIC if they are to be used after the called program terminates. All PRIVATE memory variables and arrays created within the program are released once the program terminates.

Because DO first searches for .dbo files, you should not change the file extension once a program has been compiled. All programs should have unique names, because once compiled they can no longer be distinguished by their file extensions.

If SET DEVELOPMENT is ON, DO compares the time and date stamp of a source file with the time and date stamp of its associated .dbo file. If the .dbo file is older than the source file, DO recompiles the source file before executing it.

The dBASE IV program editor, which can be accessed from MODIFY COMMAND or the Control Center, will delete an old .dbo file when a .prg file is modified, and then recompile the new .prg file (generating a new .dbo file) when you save it. As other text editors will not delete the old .dbo file and recompile a new one, SET DEVELOPMENT allows you to verify that DO does not execute an outdated .dbo file.

## Options

The WITH option allows parameters to be passed to a procedure. The parameter list can contain any valid dBASE expression, and you may pass up to 64 parameters. Field names take precedence over memory variables; so, to specify a memory variable as a parameter rather than a field with the same name, precede the memory variable name with *M->*.

## Tips

Avoid DOing a command file recursively. A command file may contain a DO command that executes itself over again, or the command file may DO a subroutine, and that subroutine may DO the original command file. Both are recursive calls, and both cases will open two files for the same command file. The error messages **Too many files are open** or **DOs nested too deep** may eventually result. dBASE IV allows a default of up to 20 DOs, but you can set this from 1 to 256 by changing the value of DO = in the CONFIG.DB file.

A procedure should return control to the calling program with a RETURN command, not with a DO command. If a command file needs to execute its own commands again, the commands should be contained in a DO WHILE loop. The command file should not DO itself again.

Using the DO command with a procedure file of any type extension produces a .dbo file. However, you should use the appropriate command for each type of file, such as REPORT FORM, to produce the correct .dbo file.

## Special Case

If the input file has a .upd extension, DO generates a .dbo file and executes the update query. You may rename this .dbo extension to .upo, if you want to keep your update query files separate from your program files.

## Examples

The following program file, Areacalc.prg, calculates the area of a rectangle based on the formula area = length * width:

```
* Program name: Areacalc.prg
PARAMETERS M_length, M_width, M_area
M_area = M_length * M_width
RETURN
* EOP: Areacalc.prg
```

To execute the program file Areacalc, pass the values 4 and 6 to the memory variables M_length and M_width respectively, and return the correct value to the memory variable called M_result:

```
. M_result = 0
           0
. DO Areacalc WITH 6, 4, M_result
Compiling line    5
        24
. ? M_result
        24
```

## See Also

CANCEL, COMPILE, CREATE/MODIFY QUERY/VIEW, DEBUG, FUNCTION, MODIFY COMMAND, PARAMETERS, PRIVATE, PROCEDURE, PUBLIC, RESUME, RETURN, SET DEBUG, SET DEVELOPMENT, SET ECHO, SET PROCEDURE, SET TRAP, SUSPEND

# DO CASE

DO CASE is a structured programming command that selects only one course of action from a set of alternatives.

## Syntax

```
DO CASE
    CASE <condition>
        <commands>
    [CASE <condition>
        <commands>]
    .
    .
    .
    [OTHERWISE
        <commands>]
ENDCASE
```

## Usage

ENDCASE terminates the DO CASE structure. Command pairs such as DO CASE...ENDCASE, IF...ENDIF, and DO WHILE...ENDDO must be properly nested within DO CASE. Nested DO CASEs are permitted.

CASE <condition> sets up a condition, or logical expression such as A = B or Numvar < 11, for evaluation. When the condition evaluates to a logical true (.T.), all subsequent commands are carried out until any one of the following commands is reached: another CASE, OTHERWISE, or ENDCASE.

After one true CASE is found and its associated commands processed, no further CASE statements are evaluated, and dBASE IV skips immediately to the first command after ENDCASE. If no CASE statements evaluate logical true, and there is no OTHERWISE statement, the program processes the first command following ENDCASE. OTHERWISE causes the program to take an alternative path of action when *all* previous CASE statements evaluate to logical false (.F.).

## Tips

Only one of the possible cases is acted upon, even if several apply. In situations where only the first true instance is to be processed, the DO CASE command is preferable to the IF command.

The CASE construct is often used when there is a small number of exceptions to a condition. The CASE <condition> statements can represent the exceptions, and the OTHERWISE statement the more common situation.

## Example

Compare this example with the example given for the IF command. The following CASE construct determines the magnitude of a variable and displays an appropriate message:

```
DO CASE
   CASE M_value > 100
      ? "Value is over 100."
   CASE M_value > 10
      ? "Value is over 10."
   CASE M_value > 1
      ? "Value is over 1."
   OTHERWISE
      ? "The value is 1 or less."
ENDCASE
```

## See Also

DO, DO WHILE, IF, IIF()

# DO WHILE

DO WHILE is a structured programming command that allows command statements between it and its associated ENDDO to be repeated as long as the specified condition is true.

## Syntax

```
DO WHILE <condition>
    <commands>
    [LOOP]
    [EXIT]
ENDDO
```

## Usage

DO WHILE <condition> opens a structured procedure that processes subsequent commands only while the condition evaluates to true (.T.): for example, .NOT. EOF() .AND. Mvar1 = 11.

If the condition evaluates to .T., all subsequent commands are carried out until an ENDDO, LOOP, or EXIT is encountered. ENDDO and LOOP return control to the DO WHILE command for another evaluation of the condition. EXIT passes control to the statement following the ENDDO.

LOOP returns control to the beginning of a DO WHILE...ENDDO program structure. LOOP prevents the execution of the remaining commands in the DO WHILE construct.

EXIT transfers control from within a DO WHILE...ENDDO loop to the command immediately following the ENDDO.

ENDDO must terminate a DO WHILE structure. The space following the ENDDO on the command line may be used for comments; the comment indicator, &&, is not needed. Comments or symbols used here will appear among compile-time warnings.

Any structured commands within a DO WHILE...ENDDO structure must be properly nested. Nested DO WHILEs are permitted.

If the condition evaluates to a logical false (.F.), dBASE IV skips all commands between DO WHILE and ENDDO and goes to the command following ENDDO.

Also note that the SCAN command, introduced with dBASE IV, offers a simple syntax for processing a series of records. SCAN has the same capabilities as DO WHILE.

## Programming Notes

You can use macros in the conditional portion of a DO WHILE loop only if the value of the variable in the macro does not change, because the DO WHILE statement is parsed only the first time through the loop. After the first parsing, the DO WHILE statement is executed from memory.

Macro substitution must also be within the lowest nested level of a program and within the lowest nested DO WHILE loop. If the DO WHILE loop that contains the macro has a nested DO WHILE loop or DO <procedure name> within it, the condition of the loop will always evaluate to logical .T. after the first evaluation, and the program will remain in an endless loop.

## Examples

The first example shows how to correctly use a macro in the conditional portion of a DO WHILE loop executing in a dBASE program file:

```
* This is an example of the correct way to use a macro in a DO WHILE statement.
*
USE Transact ORDER Client_id
Condition = [UPPER(Client_id) = "C00001"]
FIND C00001
DO WHILE &Condition. .AND. .NOT. EOF()
   * The value of Condition never changes within the loop.
   ? Order_id, Date_trans, Total_bill
   SKIP
ENDDO
CLOSE DATABASE
```

The next program file example illustrates the correct use of the EXIT option. The program lets you view five records from the Stock database file, and then decide whether you want to see the next five or back up and see the previous five.

```
* Program name: Partial.prg
USE Stock
DO WHILE .NOT. EOF()
   CLEAR
   LIST NEXT 5 Order_id, Part_name, Item_cost
   ?
   WAIT "Press X to stop, B to back up, Spacebar to continue." TO Mstop
   DO CASE
      CASE Mstop $ "Xx" .OR. EOF()
         EXIT
      CASE Mstop $ "Bb"
         SKIP -9
      OTHERWISE
         SKIP
   ENDCASE
ENDDO
* EOP: Partial.prg
```

## See Also

DO, DO CASE, IF, RETURN, SCAN

# EDIT

EDIT is a full-screen command you use to display or change the contents of a record in the active database file or view.

## Syntax

EDIT [NOINIT] [NOFOLLOW] [NOAPPEND] [NOMENU] [NOEDIT]
  [NODELETE] [NOCLEAR] [ < record number > ] [FIELDS
  < field list > ] [ < scope > ] [FOR < condition > ] [WHILE
  < condition > ]

## Defaults

If you use EDIT without a scope, or without a FOR or WHILE clause, you can move through all records in the database file.

Using a scope, or a FOR or WHILE clause, disables the **Go To** menu and the **F2** key for shifting to BROWSE.

## Usage

EDIT and CHANGE are identical commands.

You can change from EDIT to BROWSE by pressing the **F2 Data** key. EDIT displays data according to the definitions set in a format (.fmt) file if one is active, or in a default vertical field arrangement if a format is not active. BROWSE displays multiple records in a tabular format.

EDIT uses the standard full-screen cursor control keys. The arrow keys move the cursor within a record. **PgUp** backs up to the previous record. **PgDn** advances to the next record or to the next screen if the fields extend beyond one screen. To exit and save all changes, press **Ctrl-End**. Press **Esc** to exit and save changes to all but the current record. To EDIT a memo field, press **Ctrl-Home** when the cursor is positioned on the memo field name.

Unless you use the NOAPPEND option, EDIT allows you to append records to a database file if you move the cursor past the last record of the file. In this way, it works just like the full-screen APPEND command.

When called from BROWSE, EDIT respects all the BROWSE command line options except COMPRESS, FREEZE, LOCK, WIDTH, and WINDOW. BROWSE also respects all EDIT command line options.

When the EDIT command is completed, you return to your point of origin: the dot prompt or next line of a .prg file.

## Options

NOINIT allows the command line options that you used with a previous EDIT command to be used in the current EDIT. NOINIT instructs the EDIT command not to initialize the EDIT table, but to use the table from the most recent EDIT instead.

NOFOLLOW applies only to indexed data. If you specify NOFOLLOW, editing a key field in a record repositions the record to its new position in the index order; the record that then takes the old record's place becomes the current record on the screen. If you do not specify NOFOLLOW, the edited record is repositioned after the key is changed; the record following the newly-positioned record becomes the current record on the screen.

NOAPPEND prevents you from adding records to the database file during the edit.

NOMENU prevents access to the EDIT menu bar.

NOEDIT prevents you from changing any data presented on screen. You can add records, however, if you move the cursor to the end of the file, and you can mark records for deletion with **Ctrl-U**.

NODELETE prevents you from deleting records during the edit with the **Ctrl-U** key.

NOCLEAR keeps the record's image on the screen after you exit the EDIT.

<record number> starts the edit on the specified record, but lets you move to other records in the file. You may also use the keyword RECORD, which is one of the options of <scope>. If you use the <scope> keyword RECORD, however, EDIT is limited to one record, and does not allow you to move to other records in the file. Because EDIT RECORD <record number> limits the edit to the specified record, EDIT RECORD <record number> and EDIT <record number> are not identical.

## Tips

If SET AUTOSAVE is OFF, the directory entry for the active database file may not reflect all new records until the file is closed. If SET AUTOSAVE is ON, the directory on disk is updated after each new record is added.

## Special Cases

In a network environment, if you have neglected to lock a record with **Ctrl-O** or LOCK() before making a change, dBASE IV attempts to lock the record and any related records as soon as you press a key that is not a navigation key.

### See Also

BROWSE, CHANGE, CREATE/MODIFY QUERY/VIEW, MODIFY COMMAND, SET AUTOSAVE, SET DESIGN, SET FIELDS, SET FORMAT, SET LOCK, SET REFRESH, SET WINDOW OF MEMO

EJECT causes the printer to advance the paper to the top of the next page. EJECT affects only the printer. Greater functionality is available in the EJECT PAGE command.

## Syntax

EJECT

## Usage

Unless you have set the _padvance system variable to "LINEFEEDS", EJECT issues a form feed (ASCII code 12) to the printer. If _padvance is "LINEFEEDS", EJECT issues line feeds (ASCII code 10) to position to the top of form.

For proper printer operation, you must initially set the paper to the top of the form. Refer to your printer manual for instructions.

EJECT resets PROW() and PCOL() to zero.

## Tips

In a program file, you may want to verify the printer is connected and on-line with PRINTSTATUS() before issuing an EJECT.

## See Also

???, EJECT PAGE, ON PAGE, PCOL(), PRINT, PRINTSTATUS(), PROW(), SET PRINTER

Chapter 5, "System Memory Variables," includes a discussion of _padvance, _pageno, _pcolno, and _plineno.

# EJECT PAGE

EJECT PAGE either advances the streaming output to the defined ON PAGE handler on the current page, or to the beginning of the next page.

## Syntax

EJECT PAGE

## Usage

If you defined an ON PAGE handler, EJECT PAGE determines whether the current line number (_plineno) is before or after the ON PAGE line.

1. If _plineno is before the ON PAGE line, EJECT PAGE sends the appropriate number of line feeds to the output devices to invoke the ON PAGE handler.
2. If you do not have an ON PAGE handler, or if the current line number (_plineno) is after the ON PAGE line, EJECT PAGE advances the streaming output in the following manner:
   a. If SET PRINTER is ON and _padvance is "FORMFEED", EJECT PAGE sends a form feed to the printer.

      If SET PRINTER is ON and _padvance is "LINEFEEDS", EJECT PAGE sends enough line feeds to the printer to eject the current page. It calculates the number of line feeds to send to the printer with the formula (_plength − _plineno).

      If the streaming output is routed to another destination (as with SET CONSOLE or SET ALTERNATE), EJECT PAGE uses the same formula (_plength − _plineno) to determine the number of line feeds to send.

   b. It increments the _pageno system memory variable.

   c. It resets the _plineno system memory variable to zero.

## Tips

You may EJECT PAGE before the output reaches the ON PAGE line in order to call the page handler. The page handler should take care of the page eject, and may optionally write a footer on the current page and a header at the top of the next page.

Use EJECT, not EJECT PAGE, to simply eject a page on the printer. The EJECT command does not affect the _pageno and _plineno system variables, although it does honor _padvance and _pwait.

## See Also

?/??, EJECT, ON PAGE, SET ALTERNATE, SET CONSOLE, SET PRINTER, _padvance, _pageno, _plength, _plineno

Chapter 1, "Essentials," discusses system memory variables and streaming output.

# ERASE

ERASE removes a file from the disk directory.

## Syntax

ERASE <filename> /?

or

DELETE FILE <filename> /?

## Defaults

The filename must include the file extension. If the file is not on the default drive, include the drive designator.

## Usage

Use ERASE ? to display a menu of files.

You may not delete an open file.

To erase a file in another directory, you must explicitly state the path in the filename.

If you ERASE a database file (.dbf) that has memo fields, you must separately delete the .dbt file that contains the memo fields. Also be sure to delete the production .mdx file associated with a particular .dbf file.

Unlike the DOS ERASE command, dBASE IV does not permit the use of wildcard characters.

DELETE FILE is the same as ERASE.

## See Also

CLOSE, DELETE, DELETE TAG, FILE(), USE

# EXPORT

EXPORT copies the open database file to a file format usable by PFS:FILE, dBASE II, Framework II, or RapidFile.

## Syntax

```
EXPORT TO <filename> [TYPE] PFS/DBASEII/FW2/RPD
    [FIELD <field list>] [<scope>]
    [FOR <condition>] [WHILE <condition>]
```

## Usage

EXPORT creates files that can only be used by PFS:FILE, dBASE II, Framework II, or RapidFile. You should use the COPY command to create files that can be read by other software programs.

The records are exported in indexed order if an index file is in use. For PFS:FILE export, you may use a format (.fmt) file to define the screen format. If a format file is not activated with SET FORMAT, the default screen as provided in APPEND or EDIT is used to define the PFS:FILE screen format.

EXPORT creates a Framework II database frame.

If the dBASE IV database file was previously IMPORTed from PFS:FILE, it has an associated format (.fmt) file.

If a TO file already exists and SET SAFETY is ON, you are warned before the file is overwritten. If SET SAFETY is OFF, dBASE IV simply overwrites the existing file.

**NOTE**
*dBASE IV allows you to build files with fields that may be larger than your other software can accept. Although these fields are exported, they may be truncated by other programs. Check the limitations of other programs before creating files with EXPORT.*

*For example, when you EXPORT a format file to PFS:FILE, check that it does not contain more than 200 @...SAY...GET commands. Also, the form should not specify more than 21 rows, and the rows on your form must be between row 0 and row 20. PFS:FILE cannot read a file that exceeds these limitations.*

## See Also

COPY, IMPORT, SET FORMAT, SET SAFETY

# FIND

FIND searches an indexed database file for the first record with an index key that matches the specified character string or number. FIND conducts a very rapid record search.

## Syntax

FIND <literal key>

## Usage

This command positions the record pointer to the first record in an indexed database file that matches the character string or number.

FIND and SEEK both use an index, either an index (.ndx) file or multiple index (.mdx) file tag, to quickly search for data in a database file. The index used is called the *controlling* or *master* index, and it is activated with either the SET INDEX command, the SET ORDER command, or with the INDEX clause of the USE command. SEEK can search for an expression; FIND cannot.

LOCATE has a similar function to FIND and SEEK, but processes the file sequentially (record-by-record) and does not require that the file be indexed. LOCATE is generally slower.

Because FIND does not evaluate expressions in the command line the way that SEEK does, you must use a character memory variable with the &, the macro substitution function, when searching for the variable's contents:

```
. FIND &Memvar
```

Substring or partial key searches work only if the search expression matches the index key, starting with the character at the far left, and if SET EXACT is OFF. FIND will fail to locate a substring of the key if SET EXACT is ON, because it looks for an exact match for the entire length of the key. For example, FIND Smi will find "Smith" if SET EXACT is OFF, but not if SET EXACT is ON.

FIND respects the setting of SET DELETED. If SET DELETED is ON, FIND will not position the record pointer on a deleted record. FIND also ignores records blocked out by the SET FILTER command.

## Record Pointer

If a match is found, FIND positions the record pointer on the matching record.

SET NEAR affects the positioning of the record pointer after a FIND. If SET NEAR is ON (or if you have NEAR = ON in the Config.db file) and a matching record is not found, the record pointer will be on the very next indexed record in the file, just after the place where FIND expected the matching record to be. The FOUND() function will still return a false (.F.), because the key was not found; EOF(), however, will not return a true (.T.), because the record pointer is positioned to a nearly matching record in the file.

If SET NEAR is OFF, which is the default setting, and the specified character string or number is not found, the message **Find not successful** appears on the screen. SET TALK OFF suppresses this message. The record pointer moves to the end of the file, FOUND() returns false, and EOF() is true.

If another file is related with SET RELATION and the FIND is not successful, the record pointer in the related file will always be at the end of the file, whether NEAR is set ON or OFF.

The FOUND() function will only return a true for actual finds, regardless of the status of SET NEAR. The EOF() function will return a true if SET NEAR is OFF and there is no match. If SET NEAR is ON, EOF() will only return a true when the key that is sought is greater than all the keys in the index.

## Programming Notes

Because the SEEK command accepts expressions, it is normally used in program files where expressions are built by other commands or functions and passed to it. FIND is normally used for ad hoc queries from the dot prompt, although you can also use FIND in program files.

## Special Cases

FIND ignores leading blanks when searching for a literal string. The following two commands are identical:

```
. FIND A
. FIND    A
```

If you are searching for a string that contains leading blanks, include the character string in either single quote mark, double quote mark, or square bracket delimiters. You must also include the exact number of leading blanks in the string:

```
. FIND "    A"
```

If a memory variable contains leading blanks, you must enclose the macro substitution function and variable in quotation marks:

```
"&<memvar>"
```

If you are searching for a string that begins with a dBASE delimiter in the text, include the entire string within another delimiter:

```
. FIND ["Yamada"]
```

## Examples

These examples use the Client database file, indexed on the expression Lastname + Firstname. To find the first record with a Lastname beginning with the uppercase letter M:

```
. USE Client INDEX Cus_name
Master index: CUS_NAME
. FIND M
. ? Lastname
Martinez
. SET EXACT ON
. FIND M
Find not successful
. SET EXACT OFF
```

To search for a record for Paterson (there is no such record):

```
. FIND Paterson
Find not successful
. ? EOF()
.T.
. SET NEAR ON
. FIND Paterson
Find not successful
. ? EOF()
.F.
. ? FOUND()
.F.
. ? Lastname
Peters
```

## See Also

EOF(), FOUND(), INDEX, KEY(), LOCATE, LOOKUP(), MDX(), NDX(), SEEK, SEEK(), SET DELETED, SET FILTER, SET INDEX, SET NEAR, SET ORDER, TAG(), USE

# FUNCTION

FUNCTION identifies a user-defined function within a procedure.

## Syntax

FUNCTION <procedure name>

## Usage

You may create user-defined functions to perform operations on data items that cannot be accomplished by the dBASE IV functions covered in Chapter 4. When dBASE IV comes across a user-defined function in a program, it searches for a FUNCTION procedure with the same name as the user-defined function, executes the procedure, and returns a value to the command line that contained the function.

FUNCTION procedures may be contained in the current .dbo file, in a procedure file (if SET PROCEDURE is used), or in other open .dbo files. dBASE IV maintains a list of all procedures, including FUNCTION procedures, at the beginning of every object (.obj) file.

FUNCTION procedures are similar to other procedures, except that they must begin with the FUNCTION command (just as other procedures must begin with a PROCEDURE command), and they must end with a RETURN <exp> command. A call to a user-defined function passes parameters to the FUNCTION procedure, which must contain a PARAMETERS command. The FUNCTION procedure operates on these parameters, and returns values to the user-defined function through the RETURN command.

As dBASE IV searches for a FUNCTION procedure with the same name as the user-defined function, both the user-defined function and its associated FUNCTION must have the same name, but this must not be the name of a dBASE IV command or function. As with PROCEDURE, the name can be up to eight characters long and must begin with a letter.

User-defined functions have to be compiled *before* they are called from a program. They are not compiled while another program is running.

Because a user-defined function is a procedure of a special type, some dBASE IV commands, or uses of commands, are excluded:

APPEND
APPEND FROM
APPEND FROM ARRAY
APPEND MEMO
ASSIST
BEGIN TRANSACTION/
  END TRANSACTION
BROWSE
CANCEL
CHANGE
CLEAR ALL/FIELDS
CLOSE
CLOSE ALTERNATE
CLOSE FORMAT
CLOSE PROCEDURE
COMPILE
CONVERT
COPY
COPY FILE
COPY INDEXES
COPY MEMO
COPY STRUCTURE
COPY STRUCTURE EXTENDED
COPY TAG
COPY TO ARRAY
CREATE or MODIFY STRUCTURE
CREATE FROM
CREATE VIEW FROM ENVIRONMENT
CREATE/MODIFY APPLICATION
CREATE/MODIFY LABEL
CREATE/MODIFY QUERY/VIEW
CREATE/MODIFY REPORT
CREATE/MODIFY SCREEN
DEBUG
DEFINE BAR
DEFINE BOX
DEFINE MENU
DEFINE PAD
DEFINE POPUP
DEFINE WINDOW
DELETE TAG
DIR
EDIT
ERASE or DELETE FILE
EXPORT
HELP
IMPORT
INDEX
INSERT
JOIN
LABEL FORM
LOAD
LOGOUT
MODIFY COMMAND/FILE
MOVE WINDOW
ON ERROR/ESCAPE/KEY
ON PAD
ON PAGE
ON READERROR
ON SELECTION PAD
ON SELECTION POPUP
PACK
PRINTJOB/ENDPRINTJOB
PROTECT
QUIT
REINDEX
REPORT FORM
RESTORE
RESTORE MACROS
RESTORE WINDOW
RESUME
ROLLBACK
SAVE
SAVE MACROS
SAVE WINDOW
SET
SORT
SUSPEND
TOTAL
TYPE
UPDATE
ZAP

In addition to the full-screen SET command, the following SET commands cannot be included in a UDF:

| | | | |
|---|---|---|---|
| CATALOG | FORMAT | SKIP | TRAP |
| DEBUG | PROCEDURE | SQL | VIEW |
| DEVICE | RELATION | STEP | WINDOW |
| FIELDS | | | |

You may not include macro substitution (&) functions in user-defined functions, but you may include indirect file references and indirect alias references.

## Examples

The following FUNCTION, Plus_tax, returns the final cost of an item by adding the tax to the price. This user-defined function should be included in a program file, and requires that the item price and tax be passed to it as parameters.

```
FUNCTION Plus_tax
PARAMETERS M_price, M_tax
M_cost = (M_price * M_tax) + M_price
RETURN(M_cost)
* EOF Plus_tax.prg
```

The file should be available in the current directory:

```
. SET TALK OFF
. M_tax = .10
. ? Plus_tax(100, M_tax)
        110
```

## See Also

DO, PARAMETERS, PROCEDURE, RETURN, SET PROCEDURE

Chapter 1, "Essentials," contains further information on user-defined functions, including the dBASE IV commands that cannot be used in FUNCTION procedures.

# GO/GOTO

GO/GOTO positions the record pointer to a specified record in the active database file.

## Syntax

GO/GOTO BOTTOM/TOP [IN <alias>]

or

GO/GOTO [RECORD] <record number> [IN <alias>]

or

<record number>

## Usage

If an index is not in use, TOP and BOTTOM refer to the first and last records in the database file. If an index file is in use with the database, TOP and BOTTOM refer to the first and last records in the index file. GO <record number> refers to the specified record number, and not to a position in the index file. You may also position the record pointer to a specific record by issuing the numeric expression without the GO/GOTO verb.

With SET DELETED ON, you may GOTO a record that is marked for deletion by directly specifying its record number. GOTO can also move the record pointer to records that are restricted by SET FILTER, although you can't access such records with EDIT.

If a relation is set up among several files, moving the record pointer in the parent file with GOTO will reposition the record pointer in a child database file to a related record. If there is no related record, the child file's record pointer will be positioned at the end of the file, and EOF() returns a true (.T.). Moving the record pointer in a child file, however, does not reposition the record pointer in its parent file.

## Options

You may reposition the record pointer in another work area with the IN clause. The IN clause allows you to manipulate the database file in another work area without SELECTing it as the current work area. The <alias> you use may be:

- A number from 1 through 10
- A letter from A through J
- An alias name, either default or supplied through the ALIAS option of the USE command
- A numeric expression that yields a number from 1 through 10 (surrounded by parentheses if it is a simple variable)
- A character expression that yields a letter A through J or an alias name

## Programming Notes

In a network environment, the message **Relation record in use by another** appears if the related record is locked and you attempt to modify its data. You may trap error number 142 in an ON ERROR routine, and attempt another record lock.

## Examples

The following examples use the Client database file.

To position the record pointer to record five of the Client database file:

```
. USE Client
. 5
CLIENT: Record No      5
```

Use a memory variable to retain the record number:

```
. M_recno = RECNO()
               5.00
. GO TOP
CLIENT: Record No      1
. GO M_recno
CLIENT: Record No      5
```

## See Also

ON ERROR, RECNO(), SET DELETED, SET FILTER, SET RELATION, SKIP

# HELP

HELP is a menu-driven command that provides information about dBASE IV.

## Syntax

HELP [ < dBASE IV keyword > ]

## Usage

The HELP command uses the Dbase1.hlp and Dbase2.hlp files supplied with dBASE IV.

The keyword must be a dBASE IV command, function, or HELP screen name. You can press **F1 Help**, instead of typing HELP, to get the Help Table of Contents. This displays the main Help contents. Use arrow keys to move through the menu. Press ↵ to choose the highlighted option.

You can also choose to go back to reread earlier screens, print a screen of information, view related topics, or view just the syntax of a command and an example.

To exit Help, press the **Esc** key. The Help text remains on screen so you can use its information while working on the command line. The Help text disappears when you press ↵.

## Example

You can get HELP about the RUN command with:

```
. HELP RUN
```

## See Also

SET HELP, SET INSTRUCT

# IF

IF is a structured programming command that enables conditional processing of commands. The IF structure must terminate with ENDIF.

## Syntax

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

## Usage

IF is a valid command only in programs and cannot be used at the dot prompt.

Any structured commands within an IF...ENDIF structure must be properly nested. Nested IFs are permissible.

IF <condition> sets up a condition, or logical expression such as A = B or Numvar < 11, for evaluation. If the condition evaluates to a logical true (.T.), all subsequent commands are carried out until an ELSE (or the ENDIF if no ELSE exists) is reached. dBASE IV then executes the first command after ENDIF.

If the condition evaluates to a logical false (.F.), dBASE IV goes directly to the ELSE or ENDIF, whichever it encounters first. Commands between the ELSE statement and ENDIF are executed if the condition is a logical false.

If there are multiple IFs in a command structure, ELSE refers to the IF immediately preceding it in the nested structure.

The space following ENDIF on the command line may be used for comments. The comment indicator, &&, isn't needed, but you will see a compile-time warning if it is left out.

## Example

Compare this example with the example given for the DO CASE command. The following nested IF construct determines the magnitude of a memory variable and displays an appropriate message:

```
IF M_value > 100
   ? "Value is over 100."
ELSE
   IF M_value > 10
      ? "Value is over 10."
   ELSE
      IF M_value > 1
         ? "Value is over 1."
      ELSE
         ? "Value is 1 or less."
      ENDIF
   ENDIF
ENDIF
```

## See Also

DO CASE, DO WHILE, IIF(), SCAN

# IMPORT

IMPORT creates dBASE IV files from PFS:FILE forms, dBASE II database files, Framework II database and spreadsheet frames, RapidFile data files, and Lotus .WK1 spreadsheets.

## Syntax

IMPORT FROM <filename> [TYPE] PFS/DBASEII/FW2/RPD/WK1

## Defaults

The filename must include the file extension, if one exists and is different than the default extension. Usually, dBASE II database files have a .dbf extension, Framework II files an .fw2 extension, RapidFile data files an .rpd extension, and Lotus 1-2-3 (release 2.x) files have a .wk1 extension. PFS:FILE forms do not usually have an extension. The newly created dBASE IV files are given the same name as the original file, but with a .dbf extension. If you import dBASE II files, the old dBASE II database must be changed to a .db2 extension to avoid confusion with the dBASE IV file.

If a catalog is open, the new file is added to the catalog. Records created in the new database file are limited to a maximum of 4,000 bytes.

IMPORT creates a file on a drive or directory you specify with the SET PATH command. If you do not specify a drive or path, dBASE IV assumes that the file to be imported is on the default drive and directory.

## Usage

To make sure that you import PFS files correctly, check the following conditions before using the IMPORT command:

- 255 data items per form. You can have up to 255 fields in a record. (Notice, however, that headings and comments in your PFS:FILE form are also converted to fields.)
- 254 characters per data item. 254 is the maximum length for character fields in dBASE IV.

IMPORTing a PFS:FILE form creates a database (.dbf) file, a format (.fmt) file, and a compiled format (.fmo) file. All three have the same filename, and are assigned their default file extensions.

## See Also

APPEND FROM, COPY, EXPORT, SET FORMAT, USE

# INDEX

INDEX creates an index in which records from a database file are ordered alphabetically, chronologically, or numerically.

## Syntax

```
INDEX ON <key expression> TO <.ndx filename>/
    TAG <tag name> [OF <.mdx filename>] [UNIQUE]
    [DESCENDING]
```

## Defaults

Unless you specify otherwise as part of the filename, the default drive and directory are assumed. If you give a filename without an extension, the default .ndx or .mdx extension is written.

If you type only INDEX ↵ (that is, without any other keywords or options), dBASE IV prompts for the index expression, which corresponds to the ON clause, and the destination, which corresponds to the TO clause.

When SET SAFETY is ON (the default), dBASE IV displays a warning prompt before overwriting an index (.ndx) file, or a tag with the same name.

If you use TAG, but do not provide an .mdx filename with the OF clause, dBASE IV creates the tag in the production .mdx file.

Without UNIQUE, the index will index records even if they share the same key expression value.

INDEXing occurs in ascending order, unless you use the DESCENDING option. You may use DESCENDING only when building .mdx tags.

## Usage

The index, which is written to disk as either an index (.ndx) file or as a tag in a multiple index (.mdx) file, contains the key values and the corresponding record number for each record in the database file.

Indexed database files allow you to position the record pointer directly to the first record whose data matches an expression given with the FIND or SEEK commands, or with the LOOKUP() or SEEK() functions.

A multiple index (.mdx) file may contain up to 47 tags, each of which imposes an index order on the database file. Tag names follow the same naming conventions as variable names: they may be up to ten characters long, must begin with a letter, and may contain letters, numbers, and underscores. Index filenames (both .ndx and .mdx) follow the naming conventions for all filenames: they may be up to eight characters long, and may only contain characters allowed by DOS.

A production multiple index file is an .mdx file opened whenever the database file is USEd. Each database file may have one production .mdx file. The production multiple index file has the same name as the database file, but has an .mdx, rather than a .dbf, file extension. The database file header contains a flag that indicates the presence of a production .mdx file.

To create an .ndx file, specify TO <filename>. To create a tag in the production .mdx file, specify TAG <tag name>. To write the index to an .mdx file that is not the production .mdx file, specify TAG <tag name> OF <.mdx filename>.

The physical order of the records in the original database file on disk is not changed by the INDEX command. The index file contains the index key values and the corresponding record number for each record in the database file. The controlling index controls the movement of the record pointer in the database file.

Once you create an index, it becomes the new controlling index, and records will appear in the new index order. To change the controlling index, use the SET INDEX or SET ORDER command. Additional active index files have no effect on record pointer movement, and are open only so they can be updated when data in their keys are added, changed, or deleted in the database file. Whenever changes are made to a database file that affect the key, the associated index file must be open to log the changes; the alternative is to open and REINDEX the index file.

When you create an ascending index, the key expression may be a single field or any valid dBASE expression. The maximum length of the key expression is 220 characters. The maximum length of the key, the result of the evaluated index key expression, is 100 characters.

The data type of the index key expression determines whether records will be ordered chronologically (date expressions), numerically (numeric expressions), or in ASCII order (character expressions). When the index key expression includes several fields, they must all be converted to the same data type. You can use dBASE functions to convert fields to a matching type. Some of the functions most commonly used in creating index key expressions are STR(), SUBSTR(), CTOD(), DTOC(), DTOS(), YEAR(), MONTH(), DAY(), and VAL().

## Options

The UNIQUE option is the same as issuing SET UNIQUE ON before an INDEX. When several records have the same value on the key field, only the first record that dBASE IV encounters with that value is included in the .ndx file.

Whenever you REINDEX an .ndx file that was created with UNIQUE, the file retains its UNIQUE status, regardless of whether SET UNIQUE is ON or OFF.

DESCENDING will build an .mdx tag in descending ASCII order. The DESCENDING option can only refer to the entire index expression, not to one element in the expression, such as a field. You may not build an .ndx file with the DESCENDING option.

## Tips

You can specify index expressions that convert field types. For example, if you wanted to INDEX a date field chronologically and have the last names in alphabetical order for each date, you could create the following expression:

```
INDEX ON DTOS(<date>) + Last_name TO <index filename>
```

The key expression must evaluate to a fixed-length key. dBASE IV does not prohibit you from creating an index with a variable length key, but the index may not be reliable. When creating an index, always be sure to give a specific fixed length with functions that accept a length parameter, such as STR() and SUBSTR().

SORT and INDEX both reorder the records in a database file, but SORT creates a new physical database file that is written on disk. A SORT operation is analogous to COPYing an INDEXed database file.

Operations that move sequentially through the database file are usually slower if an index is open. SET ORDER TO 0 if you are not using the index to position the record pointer, but still want to update the index keys.

## Special Cases

If you use TAG, but do not provide an .mdx filename with the OF clause, dBASE IV checks the database file header for the existence of a production .mdx file. If a production .mdx file exists, dBASE IV creates the tag in that production .mdx file. If it does not find a production .mdx file, it creates one. While attempting to create the new .mdx file, it may find another .mdx file with the same name as the .mdx file it is attempting to create. In this case, it updates the database file header to show the presence of this production .mdx file, without creating a new file.

If you use the OF clause, and the .mdx file is not open, it will be opened before the index is created. If the .mdx file cannot be found, a new .mdx file is created with the filename stated in the OF clause.

If you do not use TAG, an .ndx file is created.

INDEX ignores filters set with SET FILTER or SET DELETED. All records of the database file are included in the index, even those marked for deletion or not satisfying the filter condition.

You can create an index on memory variables and fields from other open database files. Indexes that reference data not in the current database file, however, may not be updated when that data is changed.

If a catalog is open when an .ndx or .mdx file is created, the index file is added to the catalog. Index tags contained within an .mdx file are not, however, added to the catalog.

In a network environment, the database file must be in exclusive use before INDEXing. If the file is not USEd EXCLUSIVEly, the error message **Exclusive use of database is required** appears.

## Examples

To index the Transact database file by Client_id:

```
. USE Transact
. INDEX ON Client_id TO Cus_id
  100% indexed          12 Records indexed
. LIST

Record#  CLIENT_ID  ORDER_ID  DATE_TRANS  INVOICED  TOTAL_BILL
      9  A00005     87-113    03/24/87    .T.           125.00
      1  A10025     87-105    02/03/87    .T.          1850.00
     12  A10025     87-116    04/10/87    .F.          1500.00
     10  B12000     87-114    03/30/87    .F.           450.00
      2  C00001     87-106    02/10/87    .T.          1200.00
         .
         .
         .
      7  L00002     87-111    03/11/87    .F.          1000.00
```

To index a file so that it is ordered on client identifications and the amout of their transactions:

```
. INDEX ON Client_id + STR(Total_bill,10,2) TO By_amnt
  100% indexed          12 Records indexed
. LIST
TRANSACT: Record No       9

Record#  CLIENT_ID  ORDER_ID  DATE_TRANS  INVOICED  TOTAL_BILL
      9  A00005     87-113    03/24/87    .T.           125.00
     12  A10025     87-116    04/10/87    .F.          1500.00
      1  A10025     87-105    02/03/87    .T.          1850.00
     10  B12000     87-114    03/30/87    .F.           450.00
     11  C00001     87-115    04/01/87    .F.           165.00
         .
         .
         .
      7  L00002     87-111    03/11/87    .F.          1000.00
```

If you want an alphabetical list of all Client_ids, use the UNIQUE option:

```
. INDEX ON Client_id TO Clients UNIQUE
  100% indexed          7 Records indexed
. LIST Client_id

Record#  Client_id
      9  A00005
      1  A10025
     10  B12000
      2  C00001
      3  C00002
      5  L00001
      7  L00002
```

To create an index TAG of Transact in reverse chronological order:

```
. INDEX ON Date_trans TAG Recent DESCENDING
  100% indexed          12 Records indexed
. LIST

Record#  CLIENT_ID  ORDER_ID  DATE_TRANS  INVOICED  TOTAL_BILL
     12  A10025     87-116    04/10/87    .F.          1500.00
     11  C00001     87-115    04/01/87    .F.           165.00
     10  B12000     87-114    03/30/87    .F.           450.00
      9  A00005     87-113    03/24/87    .T.           125.00
      8  L00001     87-112    03/20/87    .T.           700.00
          .
          .
          .
      1  A10025     87-105    02/03/87    .T.          1850.00
```

## See Also

CLOSE, COPY INDEX, COPY TAG, DELETE TAG, FIND, KEY(), MDX(), NDX(), ORDER(), REINDEX, SEEK(), SET DELETED, SET FILTER, SET INDEX, SET NEAR, SET ORDER, SET UNIQUE, SORT, TAG(), USE

The SORT command explains the differences between INDEX and SORT in greater detail.

# INPUT

INPUT is primarily used in dBASE programs to prompt a user to enter an expression from the keyboard. Data entry is terminated by a ↵.

## Syntax

INPUT [ <prompt> ] TO <memvar>

## Usage

This command creates, if necessary, a memory variable that contains the expression entered in response to the prompt.

The prompt must be a character expression. If the prompt is a literal rather than a memory variable, it must be delimited by single quotes (' '), double quotes (" "), or square brackets([ ]).

You may enter any valid dBASE expression in response to the INPUT command. The type of expression entered determines the type of memory variable created. For instance, if you enter a number in response, a type N numeric variable is created.

You must enter a response followed by ↵. If you press ↵ without first making an entry, the prompt displays again.

## Programming Notes

If the response must be character type data, use ACCEPT, WAIT, or @...GET with the PICTURE option. ACCEPT assumes that all user response is character and doesn't require delimiters. Unlike the other commands, INPUT allows you to enter complex expressions.

If you use an INPUT command that requires a date response, include instructions in the prompt to enter the date in curly braces, which are the date delimiters. If you enter a character response, you should use the CTOD() function to convert the character variable to a date variable.

If a memory variable of the same name already exists, it is overwritten, unless you declare one variable PUBLIC and the other PRIVATE.

## See Also

@, ACCEPT, PRIVATE, PUBLIC, READ, STORE, WAIT

# INSERT

INSERT adds a single new record to the database file at the current record location.

## Syntax

INSERT [BEFORE] [BLANK]

## Usage

INSERT displays the new record for full-screen data entry. You may enter data into this record only. The new record is inserted immediately after the current record. For instance, if the current record is number 5, INSERT creates a new record 6; the old record 6 becomes record 7, and so on.

In two instances, INSERT works like APPEND and will add multiple records at the end of the file one at a time: if the file is indexed, and if the record pointer is already at the end of file.

Enter data into a memo field by placing the cursor on it (labeled **memo** on the screen) and pressing **Ctrl-Home**. Leave the memo field, by pressing **Ctrl-End** (to save) or **Esc** (to exit without saving changes). While editing the memo field, use the same control keys as MODIFY COMMAND. Press **F1** to toggle on and off the menu displaying the keys you can use.

Arrow keys move the cursor within a record. **Esc** abandons the process without inserting a new record, displaying the message **Record is not inserted**. **Ctrl-End** terminates the process and completes the record insertion.

## Options

The BEFORE clause inserts a new record just before the current record, rather than after the current record. For instance, if the current record is number 5, INSERT BEFORE creates a new record 5; the old record 5 becomes record 6, and so on.

If you include BLANK, a new record is INSERTed, but you do not enter full-screen mode. An empty record is placed in the database file. You may add data later using the BROWSE, CHANGE, EDIT, or REPLACE command.

## Tips

To copy the contents of the preceding record to the INSERTed record, use SET CARRY ON before INSERTing records.

If SET AUTOSAVE is OFF, the directory entry for the active database file may not reflect all new records until the file is closed. If SET AUTOSAVE is ON, the directory on disk is updated after each new record is added.

## Special Case

In a network environment, INSERT requires exclusive use of the database file.

## Example

To insert a new record immediately before record 4 (that is, to create a new record 4) in the Client database file, type the following:

```
. USE Client
. GO 4
CLIENT: Record No       4
. INSERT BEFORE
```

## See Also

@, APPEND, CHANGE, EDIT, MODIFY COMMAND, READ, SET AUTOSAVE, SET CARRY, SET FORMAT

# JOIN

JOIN creates a new database file by merging specified records and fields from two open database files.

## Syntax

```
JOIN WITH <alias> TO <filename> FOR <condition>
    [FIELDS <field list>]
```

## Defaults

The TO filename must include the drive and directory location if the file is not in the current directory. A .dbf file extension is assumed unless you specify otherwise.

## Usage

When JOINing an active file with an open file from an unselected work area, identify the second file by its alias name. The alias name may be the same as the filename.

The field list may consist of any type of field from both files except memo fields. If you try to join memo fields, you will get the message **Operation with memo field invalid**.

The record pointer is set to the first record in the active file. Then, each record in the second file is evaluated to see if it matches the FOR <condition>. If the specified condition is true (.T.), a new record is added to the new file. When all records in the second file are scanned, the record pointer in the active file advances to the second, and the process is repeated. This continues until all records in the active file are processed. This operation can take a long time for large files.

If you do not specify a field list, field assignments are first made from the active file. Then, fields are assigned from the second file until the 255-field limit is reached. Duplicate field names appear only once in the new file.

JOIN updates a catalog, if it is open and SET CATALOG is ON.

## Tips

Do not use the single letters A through J, or the letter M, as database filenames, because they are reserved for alias names. For example, AA is a valid database filename whereas A is not.

Make sure you have enough room when JOINing two files. Two database files can be JOINed such that the new file exceeds the available disk space. The new JOINed file can become quite large if you do not carefully choose the specified condition. For example, if two 1,000-record files were JOINed and the specified condition were always true, 1,000,000 records would be created!

## Special Cases

If both files have a field name in common and you want to include a field from the unselected work area, precede the field name with the alias name.

You may also use the SET FIELDS command before JOIN, rather than giving a field list. Only the fields listed in SET FIELDS will be included in the new database file.

## Examples

You can combine the Client database file with the Transact database file and form a new database file, Newfile, to include client and order IDs, client name, order date, and the amount of the order.

```
. USE Client
. USE Transact IN 2
```

In the following JOIN command example, Newfile is the new database file created with the fields Client_id, Client, Date_trans, Total_bill, and Order_id from the two database files, Client.dbf and Transact.dbf. Because Date_trans, Total_bill, and Order_id are found only in Transact, which is in the unselected work area, you have to let dBASE IV know where to find these fields. You can identify fields in work area 1 by the alias A or Client, or work area 2 by the alias B or Transact.

```
. JOIN WITH Transact TO Newfile FOR Client_id=B->Client_id FIELDS Client_id,
   Client, B->Date_trans, B->Total_bill, B->Order_id
12 records joined
. USE Newfile
. DISPLAY STRUCTURE

Structure for database: C:\DATA\NEWFILE.DBF
Number of data records:   12
Date of last update   : 11/23/87
Field  Field Name  Type      Width   Dec    Index
    1  CLIENT_ID   Character     6             Y
    2  CLIENT      Character    30             Y
    3  DATE_TRANS  Date          8             N
    4  TOTAL_BILL  Numeric       8     2       N
    5  ORDER_ID    Character     6             Y
** Total **                     59

. LIST NEXT 5 Client, Date_trans, Order_id

Record#  Client                   Date_trans  Total_bill  Order_id
      1  BAILEY & BAILEY          03/09/87        415.00  87-109
      2  BAILEY & BAILEY          03/20/87        700.00  87-112
      3  L. G. BLUM & ASSOCIATES  02/10/87       1200.00  87-106
      4  L. G. BLUM & ASSOCIATES  02/23/87       1250.00  87-108
      5  L. G. BLUM & ASSOCIATES  04/01/87        165.00  87-115
```

## See Also

SET FIELDS, SET RELATION

# LABEL FORM

LABEL FORM uses a specified label format file designed with MODIFY LABEL to print, display, or write labels to a file on disk.

## Syntax

```
LABEL FORM <label filename> /?
    [<scope>] [FOR <condition>] [WHILE <condition>]
    [SAMPLE] [TO PRINTER/TO FILE <filename>]
```

## Defaults

Unless the file is on the default drive or a path is set, the filename must include the drive designator and path. Unless you specify otherwise, the dBASE IV label designer gives the label design file an .lbl extension, and the generated label an .lbg extension.

Unless otherwise specified by the scope, the FOR or WHILE clause, or an active filter, labels are printed for all records in the active database file.

## Usage

The label template that you created with CREATE/MODIFY LABEL is contained in a file with an .lbl extension. The dBASE code that was generated is contained in a file with an .lbg extension. LABEL FORM finally compiles and runs an .lbo file.

LABEL FORM first determines whether the label file is in dBASE III PLUS format or dBASE IV format by checking a notation in the .lbl file. If the file is in dBASE III PLUS format, the dBASE III PLUS label engine is used to run the labels.

If the file is in dBASE IV format, and an .lbo file does not exist, LABEL FORM will compile an .lbo file and run the labels.

If an .lbo file exists and SET DEVELOPMENT is ON, LABEL FORM compares the date and time stamps of the .lbo and .lbg files. If the .lbg was created after the .lbo file, a new .lbo file is compiled before the labels are run.

If an .lbo file exists and SET DEVELOPMENT is OFF, LABEL FORM will process this .lbo without checking the stamp of an .lbg file.

## Options

Use the SAMPLE option to print test labels to ensure proper alignment of the labels in the printer. A single row of test labels is displayed using the character *x* rather than data from the database file. You may repeat the process as many times as necessary by responding Y when asked **Do you want more samples?** When you respond with N, labels begin printing using data from the file.

Use the TO FILE option to send the labels to a disk file, rather than to the printer or screen. If you direct the output to a disk file with the TO FILE option, and you have installed the ASCII text printer driver (Ascii.pr2), dBASE IV creates a file with a .txt extension. The .txt file does not contain any embedded escape codes. If you have installed any other driver, dBASE IV creates a file with a .prt extension. The .prt file may contain embedded escape sequences specific to the installed printer.

The question mark, ?, is called the query clause. Use this to activate a menu of label files to choose from.

## See Also

CREATE/MODIFY LABEL, SET PRINTER

Chapter 5, "System Memory Variables," includes a discussion of variables, such as _pdriver, which affect printed output.

*Using the Menu System* discusses the label generator in more detail.

# LIST/DISPLAY

Use LIST/DISPLAY to view the contents of a database file in an unformatted columnar list.

## Syntax

```
LIST/DISPLAY [[FIELDS] <expression list> ] [OFF]
    [ <scope> ] [FOR <condition> ] [WHILE <condition> ]
    [TO PRINTER/TO FILE <filename> ]
```

## Defaults

LIST or DISPLAY shows all records, unless limited by the scope, a FOR or WHILE clause, SET FILTER, or SET DELETED. DISPLAY shows only the current record, unless given the scope or a FOR or WHILE clause.

## Usage

LIST and DISPLAY ALL are nearly identical, except that DISPLAY pauses after each screen's display and LIST does not. After each screen, DISPLAY prompts **Press any key to continue...**

To halt a LIST or DISPLAY, press **Ctrl-S** (with SET ESCAPE ON). Press any key to resume LISTing.

To abandon a LIST, press **Esc** (with SET ESCAPE ON).

The contents of memo fields are not displayed unless explicitly named in the FIELDS list. Instead, the field name appears above the word **memo** (all lowercase) if there is no text in the memo field, or **MEMO** (all uppercase) if there is text in the memo field. If memo fields are named in the expression list, their contents are displayed in 50-character wide columns. Use the SET MEMOWIDTH command to change this default display width.

When you LIST or DISPLAY a record that is longer than 80 columns, the contents on the screen wrap around to the next line and may break up a field onto two or more lines.

## Options

TO PRINTER directs command output to the printer, and TO FILE directs the output to a file on disk.

The OFF option suppresses the record numbers.

## Special Cases

If SET HEADING is ON, each column has a heading. If the displayed item is the result of an expression involving a field (for example Cost/25), the column heading is the same as the expression.

The heading appears just as you type it. For example, to have the heading in all capital letters, type DISPLAY FIELD1. For upper and lower case headings, type DISPLAY Field1.

## Example

```
. USE Stock
. SET HEADING OFF
. LIST OFF STR(Qty,3,0) + SPACE(3) + STR(Item_cost,8,2) + " " + Part_name
     FOR Item_cost > 300

  1     1200.00  SOFA, 6-FOOT
  1      650.00  SOFA, 6-FOOT
  1     1200.00  SOFA, 8-FOOT
  1     1250.00  CHAIR, DESK
  1     1250.00  CHAIR, DESK
  1     1000.00  CHAIR, DESK
  2      350.00  CHAIR, SIDE
  1     1500.00  DESK,EXECUTIVE 5-FOOT
  1     1000.00  CHAIR, DESK
```

## See Also

SET ESCAPE, SET HEADING, SET MARGIN, SET MEMOWIDTH

# LIST/DISPLAY FILES

LIST/DISPLAY FILES displays directory information similar to that displayed by the DOS DIR command.

## Syntax

LIST/DISPLAY FILES [LIKE <skeleton>]
    [TO PRINTER/TO FILE <filename>]

## Defaults

If you do not specify a filename or skeleton, LIST/DISPLAY FILES provides directory information on database files only.

## Usage

For database files, this command displays the files, number of records, date of last update, file size (in bytes), total number of files displayed, total number of bytes for the displayed files, and the total number of bytes remaining on disk.

The directory does not reflect changes to newly changed files until after the file is closed, unless SET AUTOSAVE is on.

LIST/DISPLAY FILES is an alternate to DIR, but DIR does not accept the TO FILE or TO PRINTER options. LIST/DISPLAY FILES also allows the output to be routed to a disk file or to the printer.

## Examples

```
. LIST FILES

Database Files    # Records    Last Update    Size
CLIENT.DBF                8    11/05/87       1570
STOCK.DBF                17    11/05/87       1569
TRANSACT.DBF             12    11/05/87       554

   3693 bytes in     3 files
8357056 bytes remaining on drive
```

## See Also

DIR, FILE(), SET AUTOSAVE

# LIST/DISPLAY HISTORY

LIST/DISPLAY HISTORY outputs a list of commands that have been executed and are stored in the history buffer.

## Syntax

```
LIST/DISPLAY HISTORY [LAST <expN>]
    [TO PRINTER/TO FILE <filename>]
```

## Defaults

The default number of commands stored in the history buffer is 20. You can alter this number with SET HISTORY.

## Usage

DISPLAY pauses after each screen; LIST scrolls without pausing.

This command lets you view previously executed commands from the least recent to the most recent.

## Options

LIST/DISPLAY HISTORY displays all commands in the history buffer, unless you limit the number by including the LAST option. In this case, it displays only the last <expN> commands.

TO PRINTER directs command output to the printer, and TO FILE directs the output to a file on disk.

## See Also

SET HISTORY

# LIST/DISPLAY MEMORY

LIST/DISPLAY MEMORY provides information on how dBASE IV is using your computer's memory.

## Syntax

LIST/DISPLAY MEMORY [TO PRINTER/TO FILE <filename> ]

## Usage

DISPLAY MEMORY pauses after every screen and displays the prompt **Press any key to continue...** LIST MEMORY does not pause after each screen.

The display for numeric variables contains both the number and the type (type F or type N).

This command shows the number and names of active memory variables and elements; the public or private status of each; the values contained in each variable or element; the names, definitions, and amount of memory used by all active windows, popups, menus, and pads; the settings for all system memory variables; and the amount of memory still available.

The number of memory variables you can have is the product of the MVBLKSIZE and MVMAXBLKS settings, which are contained in the Config.db file. MVBLKSIZE is the number of memory variable *slots* each block may contain, and MVMAXBLKS is the maximum number of *blocks* in memory that may be allocated for memory variables. If MVBLKSIZE = 50 and MVMAXBLKS = 10 (the default settings), you can STORE up to 10 blocks of 50 variables each, or a total of 500 memory variables.

Memory variable blocks are allocated as they are needed. When one block is full, a second block is allocated. Additional blocks are allocated until the maximum number of blocks, which is set with MVMAXBLKS, is reached. Each memory variable slot requires 56 bytes; when the first block is allocated, dBASE reserves enough space for 50 memory variables, which is 2,800 bytes of memory.

Each array that you DECLARE takes one memory variable slot only. A special block is allocated for each array to hold the elements. Therefore, the number of array elements do not take away from the number of slots initialized by MVBLKSIZE and MVMAXBLKS.

If a memory variable is date, logical, or numeric, the data is contained within the memory variable slot. If the memory variable is character, the memory variable slot contains a pointer to another area in memory that contains the character string.

For every dBASE session, runtime symbols are created for each unique memory variable name or field name used in a program or at the dot prompt. dBASE IV creates only one runtime symbol for each field or memory variable name, no matter how many times it is referenced. The number of runtime symbols you can have is the product of the RTBLKSIZE and RTMAXBLKS settings contained in the Config.db file. RTBLKSIZE is the number of runtime symbol slots each block may contain, and RTMAXBLKS is the maximum number of blocks in memory that may be allocated for runtime symbols. The default setting for RTBLKSIZE is 50, and the default setting for RTMAXBLKS is 10, allowing you to have a total of 500 runtime symbols.

As with memory variable blocks, runtime blocks are allocated as they are needed. When one block is full, dBASE IV allocates a second block. Additional blocks are allocated until the maximum number of blocks, which is set with RTMAXBLKS, is reached. Each runtime symbol requires 17 bytes; when the first block is allocated, dBASE reserves enough space for 50 symbols, which is 850 bytes of memory.

If you run a program that references many variables or field names, you may need to allocate more memory for runtime symbols. If you have not provided for a sufficient number of runtime symbol slots in the Config.db file, the message **Exceeded maximum number of runtime symbols** appears during program execution, and you should increase the RTBLKSIZE or RTMAXBLKS settings.

A certain amount of memory overhead is used for displaying memory variables. Date variables require 8 bytes, logical variables require 1 byte, and numeric variables require 20 bytes. As the information for character variables is already stored in memory, no additional overhead is required to display these variables. The last line of LIST/DISPLAY MEMORY shows the total amount of memory overhead used to hold the display for date, logical, and numeric variables, and for any character data as stored in memory. 264 bytes are used for system memory variables; so at least 264 bytes will always show on the last line, even if you do not create any other variables. For each variable you create, the **bytes used for memvar displays & CHARS** line shows the additional overhead.

You may also decrease the Config.db settings for MVBLKSIZE, MVMAXBLKS, RTBLKSIZE, and RTMAXBLKS if you need additional memory for other operations, such as creating windows or menus.

## Options

TO PRINTER directs command output to the printer, and TO FILE directs the output to a file on disk.

## Examples

To illustrate, the character string "Hi there" has been stored to the memory variable Mgreetings; the Binary Coded Decimal 400.34 has been stored to Mfixed; a logical true (.T.) has been stored to Mtruth; and the binary 3.21E + 06 has been stored to the memory variable Mfloat.

```
. DISPLAY MEMORY
       User Memory Variables
MGREETINGS  pub   C "Hi there"
MFLOAT      pub   N           400.34 (400.3400000000000000)
MTRUTH      pub   L .T.
MFIXED      pub   F          3210000 (3210000.000000000000)
    4 out of 500 memvars defined (and 0 array elements)
    .
    .
    .
```

## See Also

DECLARE, DEFINE MENU, DEFINE PAD, DEFINE POPUP, DEFINE WINDOW, PRIVATE, PUBLIC, RELEASE, RESTORE, SAVE, STORE

Chapter 5, "System Memory Variables," includes a discussion of the system variables displayed by this command.

Chapter 6, "Customizing dBASE IV," contains information on Config.db settings, including MVBLKSIZE, MVMAXBLKS, RTBLKSIZE, and RTMAXBLKS.

# LIST/DISPLAY STATUS

LIST/DISPLAY STATUS provides information about the current dBASE IV session.

## Syntax

LIST/DISPLAY STATUS [TO PRINTER/TO FILE <filename> ]

## Usage

DISPLAY STATUS pauses after each screen's display and prompts you to **Press any key to continue...** LIST STATUS scrolls without pausing.

For each open database file, LIST/DISPLAY STATUS shows the following information:

- Current work area number
- Database name with drive, path, and alias
- Read-only status, if write-protected
- Open index filenames (both .ndx and .mdx files), index key expressions for each index file and tag, and whether the index is UNIQUE or DESCENDING
- Open memo filenames
- Filter formulas
- Database relations
- Format files

In addition, it shows:

- File search path
- Default disk drive
- Print destination
- Loaded modules
- Currently selected work area
- Left margin setting
- Currently open PROCEDURE file
- Reprocess count
- Refresh count
- The setting for DEVICE (SCREEN, PRINT, or FILE)

- Currency symbol
- Delimiter symbols
- Number of files open
- ON command settings
- Current settings for most ON/OFF SET commands
- Function key assignments

## Options

TO PRINTER directs command output to the printer, and TO FILE directs the output to a file on disk.

## Special Cases

In a network environment, LIST/DISPLAY STATUS indicates if an open database file is locked. All locked records in each work area are listed.

## See Also

ALIAS(), DIR, DISKSPACE(), FILE(), INDEX, KEY(), MDX(), MEMORY(), NDX(), ORDER(), OS(), PROGRAM(), SET, TAG(), VERSION()

# LIST/DISPLAY STRUCTURE

LIST/DISPLAY STRUCTURE displays the field definitions of the specified database file.

## Syntax

```
LIST/DISPLAY STRUCTURE [IN <alias> ]
    [TO PRINTER/TO FILE <filename> ]
```

## Usage

This command provides the following information: the filename, the number of records, the last date that any database item was changed, the complete definition of each of the fields, the fields that are index tags in the production .mdx file, and the total number of bytes in a record.

The total number of bytes in a record is the sum of all the field widths plus one (the extra byte stores the deleted record marker).

If the number of fields in the database exceeds 16, DISPLAY STRUCTURE pauses after every 16 field names and prompts you to **Press any key to continue....** LIST STRUCTURE scrolls without pausing.

If SET FIELDS is ON, the > symbol appears beside each field specified with the SET FIELDS TO command.

## Options

If you specify IN <alias> , the structure for the specified work area is displayed. The IN clause allows you to view the structure in another work area without first having to SELECT that work area.

TO PRINTER directs command output to the printer, and TO FILE directs the output to a file on disk.

## Example

To display the structure of the Client database:

```
. USE Client
. LIST STRUCTURE

Structure for database: C:\DBASE\CLIENT.DBF
Number of data records:   8
Date of last update   : 11/05/87
Field  Field Name  Type       Width   Dec   Index
    1  CLIENT_ID   Character      6             Y
    2  CLIENT      Character     30             Y
    3  LASTNAME    Character     15             N
    4  FIRSTNAME   Character     15             N
    5  ADDRESS     Character     30             N
    6  CITY        Character     20             N
    7  STATE       Character      2             N
    8  ZIP         Character     10             N
    9  PHONE       Character     13             N
   10  CLIEN_HIST  Memo          10             N
** Total **                     152
```

## See Also

COPY STRUCTURE, COPY STRUCTURE EXTENDED, CREATE or MODIFY STRUCTURE, CREATE FROM, SET FIELDS

# LIST/DISPLAY USERS

LIST/DISPLAY USERS identifies the workstations currently logged in to dBASE IV in a networking environment.

## Syntax

LIST/DISPLAY USERS

## Usage

This command reads the Login.db file in the current directory and extracts the network-assigned names of the workstations currently logged in.

## Special Cases

Always use LIST/DISPLAY USERS, or its network operating system equivalent, to determine whether anyone is using dBASE IV before it is uninstalled.

If two or more users log in with the same workstation name, this command displays the user name only once. For example, if two users log in as WKSTN1, LIST/DISPLAY USERS shows:

```
 Computer Name
> WKSTN1
```

even if they both logged in from the same directory.

## See Also

LIST/DISPLAY STATUS, NETWORK()

# LOAD

LOAD allows you to load binary program files in memory.

## Syntax

LOAD <binary filename>

## Usage

LOAD places a binary (.bin) file in memory where it can be executed with the CALL command or the CALL() function.

## Programming Notes

dBASE IV treats each loaded file as a subroutine or program module, not as an external program file.

A maximum of 16 files may reside in memory at one time. Each can be up to 32,000 bytes and must be a binary file.

Each LOADed module must have a unique name. The default extension is .bin. When you CALL a file, omit the extension; the filename itself becomes the module name.

If you LOAD a file that has the same filename but a different extension than one already LOADed, the new one replaces the first one in memory.

To see the names of LOADed modules, issue the LIST/DISPLAY STATUS command.

dBASE IV does not check the integrity of the files you LOAD. Make sure that the programs are in binary form and each program executes properly.

While designing the assembly language program from which the binary file is made, you must conform to the following specifications:

- You must originate (ORG) the first executable instruction at an offset of zero.
- The program must not allocate or use memory over and above its actual size, because LOAD uses the file size to determine how much memory to allocate.
- The program must not lengthen or shorten memory variables passed as arguments with CALL or CALL().
- Before returning control to dBASE IV, the program must restore both the Code Segment (CS) and the Stack Segment (SS) Registers.

- The program must return control to dBASE IV using a far return (RET FAR). Most commercial programs end with an exit call rather than a far return; execute these with the RUN/! command, not with LOAD and CALL.

To prepare an assembly language program written in 8086/8088 assembler for LOADing by dBASE IV, use the DOS macro assembler and the following DOS program commands:

- To assemble programs and create an object (.obj) file:

  MASM <source> <target> NULL NULL

- To link OBJ files and create an executable (.exe) file:

  LINK <target> NULL NULL

- To create a binary (.bin) file from an executable file:

  EXE2BIN <target>

## Example

Assuming that the default drive is C, execute the binary file Getdrive.bin and return the drive into the memory variable M_drive:

```
. LOAD Getdrive
. M_drive = " "
. CALL Getdrive WITH M_drive
. ? "The current drive is ", M_drive
The current drive is C
```

## See Also

CALL, CALL(), LIST/DISPLAY STATUS, RELEASE, RUN

# LOCATE

LOCATE searches the active database file for a record that matches a specified condition.

## Syntax

LOCATE [FOR] <condition> [<scope>] [WHILE <condition>]

## Usage

LOCATE performs a sequential search of a database file and tests each record for a match, usually to the FOR condition. The condition will evaluate to true (.T.) or false (.F.) for each record. If the condition evaluates to true, a match is found, and LOCATE positions the record pointer on this first matching record.

LOCATE does not require an indexed database file. FIND and SEEK, which operate on indexed files, are generally much faster. If an index is in use, however, LOCATE will follow its index order.

When an index is active, the SEEK command is more efficient. To search on the basis of a field other than the key field, close the indexes first. Because LOCATE doesn't affect data, you can SET INDEX OFF, LOCATE, and SET INDEX ON without the need to reindex.

You must provide a logical condition in the FOR clause, such as LOCATE FOR Lastname = "Goreman", or LOCATE FOR .T.

To find the next occurrence of the specified condition, use the CONTINUE command. Even if you issue several LOCATE commands against the same database file, CONTINUE always continues the search of the most recent LOCATE.

LOCATE and CONTINUE are specific to the work area in which they are issued. You can have a different LOCATE in each work area. If you issue a LOCATE, then select another work area, and later return to the first work area and issue a CONTINUE command, the search will pick up where the previous LOCATE in that work area left off.

## Record Pointer

Unless otherwise restricted by the scope or a WHILE clause, LOCATE repositions the record pointer to the beginning of the database file and starts the search with the first record.

The NEXT <n> scope option limits the search to the specified number of records. The NEXT <n> and REST scope options do not reposition the record pointer at the beginning of the database file; the search starts at the current record.

If a match is found, the record pointer moves to that record. If SET TALK is ON, dBASE IV shows the record number. FOUND() returns .T.

If no match is found, the record pointer moves to the end of the file (EOF() is .T.), or the end of the scope if you specified one, and the message **End of LOCATE scope** appears. FOUND() returns .F.

## Examples

To locate the first record in the Transact database file that contains the Client_id L00001:

```
. USE Transact
. LOCATE FOR Client_id = "L00001"
Record =       5
```

To locate each record that contains a Client_id of C00001 and has not been invoiced:

```
. LOCATE FOR Client_id = "C00001" .AND. .NOT. Invoiced
Record =       11
. DISPLAY
Record#  CLIENT_ID  ORDER_ID  DATE_TRANS  INVOICED  TOTAL_BILL
     11  C00001     87-115    04/01/87    .F.           165.00
. CONTINUE
End of LOCATE scope
. ? EOF()
.T.
. ? FOUND()
.F.
```

## See Also

CONTINUE, FIND, FOUND(), SEEK

# LOGOUT

LOGOUT logs out the current user and sets up a new log-in screen.

## Syntax

LOGOUT

## Usage

The LOGOUT command enables you to control user sign-in and sign-out procedures. The command forces a logout and prompts for a login.

When the command is processed, the workstation screen clears and a log-in screen appears. The user can then enter a group name, log-in name, and password. The PROTECT command establishes log-in verification functions and sets the user access level.

LOGOUT closes all open database files, their associated files, and program files.

## Special Cases

If PROTECT has not been used, no Dbsystem.db file is created, and LOGOUT returns the user to the dot prompt instead of to the log-in screen.

## See Also

PROTECT

# MODIFY COMMAND/FILE

MODIFY COMMAND/FILE is the dBASE IV full-screen text editor. Its function is to create and edit dBASE program and format files. You can also use it to create or edit any standard ASCII text file.

## Syntax

MODIFY COMMAND/FILE <filename> [WINDOW <window name>]

## Defaults

If you do not specify the drive, directory, or file extension as part of the filename for MODIFY COMMAND, the default drive and directory and the .prg extension are used.

MODIFY FILE, the alternative syntax for the dBASE IV text editor, does not provide a default extension. If you do not specify an extension when you create a file with MODIFY FILE, none is provided.

The default line length and right margin is 1,024 characters or column 1,024.

## Usage

MODIFY COMMAND/FILE calls up dBASE IV's full-screen text editor to create or edit program and format files. You can call it from the dot prompt or the dBASE IV Control Center, when you attempt to edit a program or procedure file, or a memo field.

MODIFY COMMAND operates in the active window. You can also assign it to an alternate window, using the WINDOW option. If the <window name> does not exist, dBASE IV displays the error message **Window name not found**.

The maximum file size MODIFY COMMAND/FILE can handle is usually limited by the amount of available disk space. It allows line lengths of up to 1,024 characters and 32,000 lines. Therefore, the maximum possible file size would be 32,000 times 1,024 characters or 32,768,000 bytes.

You can use an external word processor instead of the dBASE IV text editor, if you set it up in the Config.db file on installation. Refer to the installation instructions in Chapter 6 for the procedure to do this.

When you issue the MODIFY COMMAND/FILE command, dBASE IV searches for the specified file. If the file exists, it is called up for editing. If the file is not found, a new file is created. Use MODIFY FILE to edit an ASCII file by specifying the extension. Each time you edit a file, the previous version is saved as a backup file with the .bak extension.

The menus you can use for editing appear at the top of your screen. A complete list of the editing keys you can use is given in *Quick Reference*. More information on the use of the **Layout**, **Words**, and **Print** menus can be found in *Using the Menu System*.

You can also get a printout of a file created by MODIFY COMMAND/FILE by entering:

```
  . TYPE <filename> TO PRINT
```

## See Also

COMPILE, CREATE, DO, NOTE/*/&&, RENAME, SET DEVELOP, TYPE, UPDATE

## MODIFY Commands

The following commands are synonymous with the corresponding CREATE commands. See the CREATE command for the proper syntax and usage of these commands.

MODIFY APPLICATION

MODIFY LABEL

MODIFY QUERY/VIEW

MODIFY REPORT

MODIFY SCREEN

MODIFY STRUCTURE

# MOVE WINDOW

The MOVE WINDOW command moves a window to a new location on the screen.

## Syntax

MOVE WINDOW <window name> TO <row>,<column>/BY
    <delta row>,<delta column>

## Usage

The syntax shown combines two different ways of moving a window. You can give the new coordinates for the window, or you can give the change you want in the placement of the window, relative to its current position.

Once you move a window, the coordinates of the new location are associated with that window name. If you want the window back at its original location, you must issue another MOVE WINDOW command that contains the original row and column parameters.

If the window does not fit on the screen at the new location, an error message appears and the MOVE WINDOW command does not take effect.

## Examples

To move a window called W1 to the right by 2 columns and down by 5 lines:

```
. MOVE WINDOW W1 BY 5,2
```

To move the window W1 to new coordinates:

```
. MOVE WINDOW W1 TO 10,5
```

## See Also

ACTIVATE WINDOW, DEACTIVATE WINDOW, DEFINE WINDOW, RESTORE WINDOW, SAVE WINDOW, SET WINDOW OF MEMO

# NOTE/*/&&

NOTE, an asterisk (*), or double ampersand (&&) characters indicate comment lines in a program (.prg) file.

## Syntax

NOTE/* <text>

and

[<command>] && <text>

## Usage

NOTE/*/&& inserts comments into program files for documentation and explanatory purposes. Use NOTE and * at the beginning of a line within a program. Use && to insert comments on the same line with a command in a program. You may use the && either after the command and before the comment, or at the beginning of a line.

If a NOTE or && line ends with a semicolon, dBASE IV reads the next line as part of the comment line. You cannot use a semicolon with NOTE/*. Each line must begin with either NOTE or *.

## Examples

This example shows how you might use NOTE:

```
NOTE This is a simple loop
STORE 1 to Cnt
DO WHILE Cnt < 100
   STORE Cnt + 1 TO Cnt
ENDDO
```

This example uses && to put a comment on a program line:

```
Memvar = 12      && initializes numeric memvar
```

## See Also

MODIFY COMMAND/FILE, PROCEDURE

# ON ERROR/ ESCAPE/KEY

ON ERROR/ESCAPE/KEY calls the execution of a program if an error occurs, the **Esc** key is pressed, or the < key label name > key is pressed.

## Syntax

```
ON ERROR <command>
    /ESCAPE <command>
    /KEY [LABEL<key label name>] [<command>]
```

## Usage

The ON command sets a trap that waits for the specified condition to occur. These conditions are a dBASE IV error, pressing the **Esc** key, or pressing either a key designated in < key label name > or any key if no key label is designated. dBASE IV errors include, for example, syntax errors and evaluation errors. The ON condition remains in effect until you explicitly remove it by entering ON ERROR/ESCAPE/KEY with no command, or until you leave dBASE IV. Note that ON ESCAPE does not work when SET ESCAPE is OFF.

If ON ERROR is triggered, and you are executing the ON ERROR command or procedure, the ON ERROR trap is disabled until the command or procedure completes its execution. You may, however, set another ON ERROR trap inside the procedure. Therefore, you can nest several ON ERROR traps, each of which is released when the called command or procedure is finished.

If the < command > you give is to run another dBASE program or user-defined function, be sure that the program or UDF has already been compiled. dBASE doesn't compile external program modules while running a program.

## Programming Notes

dBASE IV responds to the ON options in the following order of precedence:

| | | |
|---|---|---|
| ON ERROR | = | A dBASE IV error occurred |
| ON ESCAPE | = | The **Esc** key is pressed |
| ON KEY | = | The specified key is pressed (or, if no key label is given, any key is pressed) |

If, for example, ON KEY (without a key label) and ON ESCAPE are in effect at the same time, pressing the **Esc** key executes the ON ESCAPE command. It does not execute the ON KEY command line.

If you try to trap the **Esc** key with the ON KEY command, SET ESCAPE must be OFF.

If the ON KEY command is in effect without a key label, and you press any key other than **Esc**, ON KEY is executed after the current command is completed. For instance, pressing a key while INDEXing a file will not interrupt the index procedure. Unless the program removes the stored key, however, the command you specify on the ON KEY command line will execute repeatedly.

ON ERROR does not respond to errors at the operating system level, such as a drive or printer not being accessible. It responds only to dBASE IV errors, such as a syntax error or an evaluation error. Also, returning from an ON ERROR <command> will not re-evaluate an IF or DO WHILE condition in the program in which the error occurred (unless RETRY is used).

The ON ERROR and ON ESCAPE commands can in turn execute any other commands, provided they aren't already being used as descendents of a user defined function or the ON KEY, ON READERROR, or ON PAGE commands.

The ON KEY command is prohibited from using the same commands that are excluded from user defined functions. See the FUNCTION command for the list of exclusions.

The LABEL option of the ON KEY command makes it possible to trap specific keys, rather than any key as in dBASE III PLUS. If you use a character memory variable for the <key label name>, include the macro substitution symbol (&). Using ON KEY LABEL without <key label name> sets the trap for any key on the keyboard. Using ON KEY alone resets the key trap and turns it off.

When SET FUNCTION and ON KEY are both sensitive to the same key, ON KEY takes precedence over SET FUNCTION. If ON KEY traps a key while you are typing input for a GET variable, the command for the trapped key will be executed and processing returns to the point prior to the command. Please note, however, that the trapped character will not be placed in the GET variable.

While a WAIT command is active, it takes precedence over ON KEY. The memory variable used by WAIT will receive its input regardless of any ON KEY traps.

ON KEY causes an immediate trigger in EDIT, BROWSE, READ, user-defined menus and popups, and at the end of each command. You can trap both control and non-control keys. Certain commands with specific tasks such as LIST, SORT, and INDEX aren't interrupted by an ON KEY press; any ON KEY commands are executed after these commands are completed.

ON KEY isn't case-sensitive regarding the <key label name> used in a command, whether for printing or non-printing characters. You can use upper- or lower-case characters.

The <key label name> for printable keys are the characters, digits, or symbols you see. Table 2-5 shows you the <key label name> descriptions to use for the non-printing keys of your keyboard.

Table 2-5 < key label name > descriptions for non-printing keys

| **Keycap Identification** | **< key label name >** |
|---|---|
| **F1** to **F10** | F1, F2, F3 . . . |
| **Ctrl-F1** to **Ctrl-F10** | Ctrl-F1, Ctrl-F2 . . . |
| **Shift-F1** to **Shift-F9** | Shift-F1, Shift-F2 . . . |
| **Alt**-0 to **Alt**-9 | Alt-0, Alt-1, Alt-2 . . . |
| **Alt**-A to **Alt**-Z | Alt-A, Alt-B, Alt-C . . . |
| ← | LEFTARROW |
| → | RIGHTARROW |
| ↑ | UPARROW |
| ↓ | DNARROW |
| **Home** | Home |
| **End** | End |
| **PgUp** | PgUp |
| **PgDn** | PgDn |
| **Del** | Del |
| **Backspace** | BACKSPACE |
| **Ctrl-←** | Ctrl-leftarrow |
| **Ctrl-→** | Ctrl-rightarrow |
| **Ctrl-Home** | Ctrl-Home |
| **Ctrl-End** | Ctrl-End |
| **Ctrl-PgUp** | Ctrl-PgUp |
| **Ctrl-PgDn** | Ctrl-PgDn |
| **Ins** | INS |
| **Tab** | TAB |
| **Back Tab** | BACKTAB |
| **Ctrl**-A to **Ctrl**-Z | Ctrl-A, Ctrl-B, Ctrl-C . . . |

## Examples

The examples below assume that a procedure file containing IF_err and Stop is already open.

To activate the ON ERROR option when a dBASE IV error occurs, type the following:

```
. ON ERROR DO If_err
```

Then, if dBASE IV encounters an error, it branches to:

```
PROCEDURE If_err
<If_err.prg command list>
.
.
* Turn off the on error flag if you
* don't want to trap additional errors.
ON ERROR
RETURN
```

To set up the ON KEY option to execute a DO <program> command that halts printing and branches to a procedure named Stop, enter:

```
. ON KEY DO Stop
```

Afterward, if you press any key, the program branches to the Stop procedure. The following procedure prompts you to press the letter **A** if you want to stop printing. Pressing any other key causes printing to resume.

```
PROCEDURE Stop
* First clear the key that triggered ON KEY
* from the type-ahead buffer with INKEY().
i = INKEY()
WAIT "Press A to Abort printing, " +;
     "any other key to continue" TO choice
IF UPPER(choice) = "A"
   RETURN TO MASTER
ENDIF
RETURN
```

## See Also

INKEY(), LASTKEY(), ON READERROR, PROCEDURE, READKEY(), RETRY, RETURN, SET ESCAPE, SET PROCEDURE, UPPER(), WAIT

# ON PAD

ON PAD associates a pop-up menu with a pad of a given bar menu.

## Syntax

```
ON PAD <pad name> OF <menu name> [ACTIVATE POPUP
    <popup name>]
```

## Usage

The pop-up menu named with the ON PAD command is activated whenever the cursor is on the pad of the specified bar menu.

If you do not use the ACTIVATE POPUP option with the ON PAD command, the prompt pad is disabled.

By using the ACTIVATE POPUP option, you can have both bar menus and pop-up menus active on the screen at the same time. As you use the → and ← keys to move within a bar menu, you activate different pop-up menus. You can use the ↑ and the ↓ keys to move between the BARS of the pop-up menus.

You cannot use this command if you have already used the ON SELECTION PAD command to associate a prompt or a program with a pad.

## Example

Please see the example provided for the DEFINE PAD command.

## See Also

DEFINE MENU, DEFINE PAD, DEFINE POPUP, ON SELECTION PAD

# ON PAGE

ON PAGE is the dBASE IV command for triggering an action when the streaming output reaches a particular line on the current page during a ?/?? or EJECT PAGE command. A typical use of ON PAGE is handling page breaks with footers and headers while printing a report.

## Syntax

ON PAGE [AT LINE <expN> <command>]

## Usage

ON PAGE executes the specified command when dBASE IV has passed the line number designated in the AT LINE clause while executing a ?/?? or EJECT PAGE command. dBASE IV keeps track of the line number by updating the _plineno system variable (see Chapter 5, "System Memory Variables") while printing is in progress.

ON PAGE permits a program to execute a valid command, such as a footer and header procedure, at the end of each page. The command executed by ON PAGE is known as the *page handler.*

ON PAGE with no argument disables the page handler.

To determine the value of <expN> for the AT LINE clause, use this formula: <expN> = page length − bottom margin − footer height.

The page length is the value of the _plength system variable. The footer height is the number of lines in the footer, as defined by the footer procedure (see "Example," below). The bottom margin is the number of blank lines below the last line of the footer.

dBASE IV keeps track of the number of lines in the header, and adjusts the lines of text on that page accordingly. You must ensure, however, that the number of lines in the footer don't exceed the lines remaining on the page. If they do, the footer will run onto the top of the next page.

If you have used the report generator to create a report with headers and footers, the REPORT FORM command activates the page handler automatically as it prints the report. You can also use a page handler with other commands that produce printed output, such as DISPLAY or SCAN.

Each of your footer procedures must begin with a ? command to replace the line feed pre-empted by ON PAGE activation. Be sure to end footer procedures with EJECT PAGE. Other procedures that involve the use of report bands should begin with a ? command and end with a ?? command.

The ON PAGE command is prohibited from using the same commands that are excluded from user-defined functions. See the FUNCTION command for the list of exclusions.

If the <command> you give is to run another dBASE program or user-defined function, be sure that the program or UDF has already been compiled. dBASE doesn't compile external program modules while running a program.

## Example

Assuming you want six-line top and bottom margins, a program file to print an inventory might look like this:

```
* Name...: Invntry.prg
ON PAGE AT LINE 60 DO Page_brk
SET TALK OFF
SET PRINT ON
DO Header                  && Print first page header.
SCAN ALL
   ? PART_NO, DESCRIPT, ON_HAND  && Print 3 fields.
ENDSCAN
EJECT PAGE                 && Activate the ON PAGE handler.
DO Footer                  && Print last page footer.
SET PRINT OFF
ON PAGE                    && Disable the page handler.
SET TALK ON
RETURN
* EOP: Invntry.prg
PROCEDURE Header
EJECT PAGE                 && Start on a new page.
?                          && Print the heading on line two.
? "Current Inventory" STYLE "B", DATE() AT 70
?
?                          && Start LISTing on the seventh line.
RETURN
* EOP: Header
PROCEDURE Footer
?
?                          && Print the footer on line 63.
? "CONFIDENTIAL - Do not distribute!" STYLE "B"
?? "Page " AT 70, LTRIM(STR(_pageno,4,0))
RETURN
* EOP: Footer
PROCEDURE Page_brk
DO Footer                  && Print a mid-report footer.
DO Header                  && Print a mid-report header.
RETURN
* EOP: Page_brk
```

## See Also

PRINTJOB, PROCEDURE, REPORT FORM, SET PRINTER, _plength, _plineno

# ON READERROR

Use ON READERROR for error trapping and recovery during full-screen operations.

## Syntax

ON READERROR [ <command> ]

## Usage

Use ON READERROR to activate a command, program, or procedure after checking for an error condition. The errors trapped by ON READERROR are invalid dates, a RANGE specification during data entry that is out of range, or an unmet VALID <condition>. When your program encounters these, it will execute the command or program specified in the ON READERROR command line. Specifying ON READERROR without a command disables the program's ability to trap errors.

The ON READERROR command is prohibited from using the same commands that are excluded from user defined functions. See the FUNCTION command for the list of exclusions.

## Tips

This command will usually be a DO <command file> to recover from the error or send a help message to the screen. You can prompt for the correct input by having the program specify valid entries.

When the <command> you give is to DO another dBASE program, be sure that the program has already been compiled. dBASE doesn't compile external program modules while running a program.

## See Also

@, APPEND, CHANGE, EDIT, INSERT, ON ERROR, READ

# ON SELECTION PAD

ON SELECTION PAD associates an action with a bar menu pad so that selecting that bar menu pad executes the specified command, procedure, or program.

## Syntax

ON SELECTION PAD <pad name> OF <menu name> [<command>]

## Usage

When you select any pad from an active menu, ON SELECTION PAD executes what has been specified in the <command> clause. The <command> clause can contain a dBASE IV command or a DO <program name/procedure name> instruction. A procedure associated with ON SELECTION PAD can use functions such as MENU() or PAD() to determine the active menu name and the last selected pad, and to perform appropriate actions.

After a command, procedure, or program is finished, the menu is available again for another selection.

If you use the ON SELECTION PAD command without a command or program, the named pad is disabled.

The ON SELECTION PAD command can in turn execute other commands, provided ON SELECTION PAD isn't already being used as a descendent of a user defined function or the ON KEY, ON READERROR, or ON PAGE commands.

## Example

In the sample files, the Exit_pop menu is activated when ↵ is pressed on the Exit pad of the Main bar menu. This is accomplished with the command:

```
. ON SELECTION PAD Exit OF Main ACTIVATE POPUP Exit_pop
```

## See Also

DEFINE MENU, DEFINE PAD, MENU(), ON PAD, PAD(), PROMPT()

# ON SELECTION POPUP

ON SELECTION POPUP specifies the command or procedure that executes when a pop-up menu selection is made.

## Syntax

ON SELECTION POPUP <popup name> /ALL [ <command> ]

## Usage

Use this command to associate any one of the prompts of the specified pop-up menu with a dBASE IV command or procedure. Use the DO command to execute a program or a procedure.

If you use ON SELECTION POPUP without specifying a command, the active popup is disabled. If you use ALL for the pop-up name, this command applies to all of the popups.

User-defined menu commands should be issued in the following order:

1. DEFINE POPUPS.
2. ON SELECTION POPUP.
3. ACTIVATE POPUP.

When you select any one of the prompts from the pop-up menu by pressing ←┘, the menu is deactivated while the command or the procedure associated with ON SELECTION POPUP executes.

The ON SELECTION POPUP command can in turn execute other commands, provided ON SELECTION POPUP isn't already being used as a descendent of a user defined function or the ON KEY, ON READERROR, or ON PAGE commands.

## Examples

When ←┘ is pressed on the Exit_pop menu, the PROCEDURE Exit_pro executes:

```
. ON SELECTION POPUP Exit_pop DO Exit_pro
```

Typically, a procedure executed from a menu will contain both a DO CASE structure to evaluate the user's selection and which operation to perform in response to the selection. The POPUP(), PROMPT() and BAR() functions may be used to identify the user's selection.

Here is a sample procedure for the View_pop menu:

```
DO CASE
   CASE BAR()=1
      APPEND BLANK
      EDIT NEXT 1
   CASE BAR() = 2
      EDIT NEXT 1
   CASE BAR() = 4
      DELETE
ENDCASE
RETURN
```

## See Also

BAR(), DEFINE BAR, DEFINE POPUP, POPUP(), PROMPT()

# PACK

PACK removes records that are marked for deletion from the active database file.

## Syntax

PACK

## Usage

All open index files are automatically REINDEXed.

After you execute a PACK command, the disk space used by the deleted records is reclaimed when the file is closed.

Note that the DIR and LIST/DISPLAY FILES commands do not accurately reflect increases or decreases in file sizes until the file is closed, if SET AUTOSAVE is OFF.

## See Also

COPY, DELETE, DELETED(), DIR, RECALL, REINDEX, SET AUTOSAVE, ZAP

# PARAMETERS

PARAMETERS assigns local variable names to data items passed from a calling program. This command receives variables passed using the command DO <filename> WITH <parameter list>.

## Syntax

PARAMETERS <parameter list>

## Usage

When included in a dBASE program file, PARAMETERS must be the first executable command. In a procedure file, PROCEDURE <filename> is the first command, followed by the PARAMETERS command. The parameters you pass can be any legitimate expressions. The parameter list assigns local variable names to receive parameters from the sending program's parameter list. The local variables created by specifying PARAMETERS are discarded when control is returned to the calling program. The parameter list must contain the same number of items as in the calling program. You may pass up to 50 parameters from the calling program.

If the parameter being passed is a memory variable, its value may be changed. The change is transferred to the variable in the calling program. If the parameter being passed is an expression, the value of that expression is stored to a memory variable in the receiving program.

## Examples

The following command file calculates the area for a given length and width:

```
*The command file Areacalc.prg
PARAMETERS Length, Width, A
* A corresponds to area
A = Length * Width
RETURN
*EOF:  Areacalc.prg
```

To execute this command file:

```
. area = 0
          0
. DO Areacalc WITH 6, 4, area
                 24
. ? area
         24
```

## See Also

@, DO, PRIVATE, PROCEDURE, PUBLIC, SET PROCEDURE, STORE

# PLAY MACRO

This command executes macros from the current macro library.

## Syntax

PLAY MACRO <macro name>

## Usage

A macro is a series of keystrokes that you record to a slot in the current macro library. You give each macro a unique identifier, and you can execute the keystrokes at any time by invoking the identifier.

When you create a macro from the Control Center, you identify it with a macro key and a macro name. Outside the Control Center, the macros you create are identified by their macro key.

The macro key may be **Alt-F1** through **Alt-F9**, or it may be **Alt-F10** followed by a letter from A to Z. Therefore, you can assign up to 35 keys, and create up to 35 unique macros at one time. The macro name may be up to 10 characters long, and must not include spaces.

The macros that you create and save can be played back in three ways:

- By pressing **F10**, then choosing **Play** from the **Tools** menu of the Control Center.
- By typing the macro key from the keyboard. You may press **Alt-F1** through **Alt-F9**, or **Alt-F10** followed by a letter from A through Z (case does not matter).
- By using the PLAY MACRO command with the macro name.

As PLAY MACRO allows you to execute the macro by name, you can include this command in program files to execute a macro without prompting the user to type the macro key.

PLAY MACRO has two important characteristics when used within programs. First, a macro is held in memory until there is an opportunity to simulate input. Using PLAY MACRO at the dot prompt gets an immediate result because the dot prompt is always ready for input, whether real-time or recorded. In a program, macro keystrokes aren't delivered until another command creates an opportunity to input those keystrokes. An example would be to give the PLAY MACRO command, and to follow it with APPEND so the keystrokes could be played into a record.

In a program, a series of PLAY MACRO commands are stacked on a "last in, first out" basis (LIFO). For example, if you write a program that has three PLAY MACRO commands in a row, the third macro will be played first, and the first macro last.

## See Also

RESTORE MACROS, SAVE MACROS

# PRINTJOB/ ENDPRINTJOB

PRINTJOB/ENDPRINTJOB are structured programming commands that control a printjob.

## Syntax

```
PRINTJOB
 <commands>
ENDPRINTJOB
```

## Usage

Use PRINTJOB to activate printjob-related settings which you define with system memory variables.

A system memory variable is a memory variable initialized by dBASE IV which you can modify. See Chapter 5, "System Memory Variables," for a description of each.

PRINTJOB causes dBASE IV to:

- Send the starting print codes defined by the _pscodes system variable
- Eject the paper if _peject is set to either "BEFORE" or "BOTH"
- Initialize _pcolno to 0

ENDPRINTJOB causes dBASE IV to:

- Send the ending print codes defined by the _pecodes system variable
- Eject the paper if _peject is set to either "AFTER" or "BOTH"
- Loop back to PRINTJOB the number of times defined by _pcopies

In your code, place the definitions for these system variables before the PRINTJOB command. If necessary, redefine any system variables used in your code after ENDPRINTJOB.

You can define other system variables, as well, for a specific printjob. If you want a feature to apply to an entire printjob, define the variable before the PRINTJOB command. For example, you might want to pause the printing after each page or define a specific page offset.

You can use PRINTJOB and ENDPRINTJOB only within a program, and you cannot nest printjobs.

## Examples

The following example sets up a custom printjob using the Client.dbf database example file.

```
* Customer.prg
USE Client
_peject = "AFTER"
SET PRINT ON
PRINTJOB
ON PAGE AT LINE _plength-1 DO PageBrk
SCAN
   ?? Client_id AT 0,Client AT 8
   ?
ENDSCAN
ENDPRINTJOB
SET PRINT OFF
RETURN
* EOP: Customer.prg

PROCEDURE PageBrk
EJECT PAGE
?? DATE() AT 1, "Page:" AT 67, _pageno PICTURE "999" AT 73
?
?
RETURN
* EOP: PageBrk
```

The following program code prints three copies of a report from page ten to the end, and ejects a page after the printing. The REPORT FORM command that runs the report contains the PRINTJOB construct so you don't need to issue the PRINTJOB/ENDPRINTJOB commands when using REPORT FORM. This routine resets _pbpage and _pcopies to their default values after the printjob executes.

```
_pbpage = 10
_peject = "AFTER"
_pcopies = 3
REPORT FORM Accounts TO PRINT
_pbpage = 1
_pcopies = 1
```

## See Also

ON PAGE, PROCEDURE, REPORT FORM, RETURN, SET PRINTER, SET TALK, _pcopies, _pecodes, _peject, _plineno, _pscodes

# PRIVATE

PRIVATE allows you to create local memory variables in a lower-level program with the same names as memory variables that were created in a calling program or were previously declared as PUBLIC.

## Syntax

```
PRIVATE ALL [LIKE/EXCEPT <skeleton>]
```

or

```
PRIVATE <memvar list>
```

## Usage

Changes made to a PRIVATE memory variable do not overwrite the value of a memory variable with the same name created in another program. Previous values of PRIVATE memory variables are retained in memory with the names of the programs that created them.

When the program containing PRIVATE memory variables ends, the values that were hidden by PRIVATE are reinstated. The PRIVATE values are no longer in effect.

When you declare a system memory variable in a dBASE IV procedure, a PRIVATE copy is created and it is assigned the current value for that system variable.

## See Also

DECLARE, DO, PARAMETERS, PUBLIC, STORE

# PROCEDURE

PROCEDURE identifies the beginning of a subroutine.

## Syntax

PROCEDURE <procedure name>

## Usage

A procedure is a useful subroutine that you run with the command DO <procedure>. You can incorporate procedures directly into a program file, or put them into a separate procedure file. dBASE IV will search for the procedure in the following order:

1. Look in the current object file.
2. Look in the SET PROCEDURE file.
3. Look in other open object files, in most-recently-opened order.
4. Look for an object file with the procedure name, and open it.
5. Look for a program file with the procedure name, and compile it.
6. Look for an SQL program file with the procedure name, and compile it.

You can repeat procedure names, because the above search pattern can effectively hide one procedure in favor of another that is higher up in the search path.

You can put as many procedures in a program or procedure file as available RAM permits, up to a maximum of 1,170 procedures per file. Each procedure must begin with the keyword PROCEDURE and end with RETURN. When a procedure begins with a PROCEDURE command, you cannot include an expression in the RETURN command line.

Procedure names may be up to eight characters long. They may contain letters, numbers, and underscores, and must begin with a letter. You should not assign the same name to a program file and to a procedure contained in the program file.

## Programming Notes

dBASE IV maintains a procedure list at the beginning of every object (.dbo) file. It treats the main program itself as a procedure, giving it a procedure name that matches the source program filename. This procedure name is the first one in the procedure list. Each subsequent procedure, whether contained in the current source file or in a separate procedure file, is added to the list when the source file is compiled to an object file.

For example, suppose you have the following program, Main:

```
*Main.prg
<commands>
DO A
DO B
DO C
RETURN

PROCEDURE A
<commands>
RETURN

PROCEDURE B
<commands>
RETURN

PROCEDURE C
<commands>
RETURN
```

dBASE IV will include four procedures in the procedure list for the compiled object file: Main (the default procedure name), A, B, and C. Note that only code at the beginning of a program file is assigned the default procedure name.

Putting the procedures in the main program file eliminates the need for many separate program files. This makes programs run more quickly, since dBASE IV does not have to open and close these programs before running them. You can still use the SET PROCEDURE TO <procedure file> command for procedures to be activated with the DO command.

Any procedure found in an active object file is available for the DO <procedure name> command. If A.dbo calls B.dbo calls C.dbo, all the procedures defined within A, B, and C are available to any procedures in C.

## Examples

The following file contains two routines which are used to output specific messages to the screen:

```
* This is an example of the procedure file Procl.prg

PROCEDURE Messagel
   @ 15,0 CLEAR
   @ 18,0 SAY "The account is now overdrawn"
RETURN

PROCEDURE Message2
   @ 10,0 CLEAR
   @ 15,10 SAY "Payment has not been received"
   @ 17,10 SAY "Please issue account hold"
RETURN
* EOP: Procl.prg
```

To activate a procedure within Proc1.prg:

```
. SET PROCEDURE TO Procl
. DO Messagel
```

## See Also

COMPILE, DEBUG, DO, PARAMETERS, RETRY, RETURN, SET PROCEDURE

# PROTECT

PROTECT creates and maintains security on a dBASE IV system.

## Syntax

PROTECT

## Usage

This menu-driven command is issued within dBASE IV by the database administrator, who is responsible for data security. PROTECT works in a single-user or multi-user environment.

PROTECT is optional. If you use it, however, the security system always controls database file access.

This command displays a full-screen menu. The first time you use PROTECT, the system prompts you to enter and confirm an administrator password.

---

**WARNING**
Remembering the administrator password is essential. You can access the security system only if you can supply the password. Once established, the security system can be changed only if you enter the administrator password when you call PROTECT. Keep a hard copy of the database administrator password in a secured area. There is no way to retrieve this password from the system.

---

If you intend to use SQL, you must add the super user log-in name SQLDBA, which is granted privileges to all operations in SQL mode. The SQL GRANT and REVOKE commands control file and field access privileges using log-in names assigned by PROTECT.

PROTECT includes three distinct types of database protection:

- *Log-in security*, which prevents access to dBASE IV by unauthorized personnel
- *File and field access security*, which allows you to define what files, and fields within files, each user can access
- *Data encryption*, which encrypts dBASE files so that unauthorized users cannot read them

Table 2-6 summarizes the database security types, how to implement each security type, and the results of security implementation.

Table 2-6 dBASE security summary

| **Security Type** | **You Define:** | **You Get:** |
|---|---|---|
| Log-in | User name and password | Control over access to dBASE IV |
| File and Field Access | Access levels | Control over access to data files, fields in data files, and application code |
| Data Encryption | User and file group | Automatic encryption and decryption of data |

It is not necessary to implement all three levels of security; you can stop at the log-in level, if you wish. You must implement the security types in the order shown in Table 2-6.

Log-in security is the first security level. Once a security system is in place, users cannot access dBASE IV until they pass log-in security.

Access control is the next security level. Access control determines what a user can do both with a database file and the data in the file, and can be used to control processing of application code. *User access levels* are numbered 1 through 8, where 1 has the greatest and 8 the lowest access privileges. You establish an access level for each user in the user's profile, and additional access levels for file and field privileges in the *file privilege scheme*.

You establish privileges for a database file by assigning access levels, in any combination, for read, update, extend, and delete operations.

Data encryption scrambles the database so that unauthorized users cannot read the information in the file.

## The Dbsystem.db File

PROTECT builds and maintains a password system file, called Dbsystem.db, which contains a record for each user who accesses a PROTECTed system. Each record, called a *user profile*, contains the user's log-in name, account name, password, group name, and access level. When a user enters the dBASE command at a network workstation, dBASE IV looks for a Dbsystem.db file. If it finds this file, it initiates the log-in process. If it does not find this file, there is no log-in process.

Dbsystem.db is maintained as an encrypted file. You will probably want to keep a hard copy record of some or all of the information contained in Dbsystem.db. For example, if a user forgets a log-in value (such as group, log-in name, or password), you will want to have that information available.

The **Reports** menu allows you to display or print security information about users and files.

### File Privileges

You can create file privilege schemes for up to eight database files at a time. If you try to set up a ninth file, you will get the error message **Too many files are open**. When you have finished creating the eighth scheme, you must store and save these eight privilege schemes before creating any more.

File access rights cannot override a read-only attribute established for the file at the operating system level.

### Changing a File Privilege Scheme

When you select a file, PROTECT checks to see if the file privilege scheme has already been defined. If it has, the rest of the menu items are completed with their current values. You can then change any of the values. If the file privilege scheme has already been saved, the message **<filename>.crp already exists, overwrite it? (Y/N)** appears. Press Y if you wish to change any values. (This is not affected by the status of SET SAFETY.)

After you make your changes, store and save them.

### Exiting from PROTECT

The **Exit** menu contains three options:

Select **Save** to post all new and updated user profiles and file privilege schemes that have been stored during the current PROTECT session. User profiles are saved in the current Dbsystem.db file. File privilege schemes are saved in the database file structure. You can save user profiles at any point during a PROTECT session. You must save file privilege schemes after you define or change eight of them. Database files are encrypted when a file privilege scheme is saved.

Select **Abandon** to cancel all new and updated user profiles and file privilege schemes not already saved during the current PROTECT session.

Select **Exit** to terminate the current PROTECT session. New and updated user profiles and updated file privilege schemes will be encrypted and saved, if they have not already gone through this process.

### Data Encryption

If your database system is PROTECTed, dBASE IV automatically encrypts and decrypts database files and their associated index and memo files.

When a database file's privilege scheme is saved, PROTECT creates an encrypted version of the database file with a .crp extension. You may want to copy the unencrypted file, for reference, to a floppy disk and store the floppy disk in a secure place. To use MODIFY STRUCTURE, you need unencrypted versions of a database file.

To enable security, you should:

1. Copy the encrypted file (.crp) over the unencrypted file (.dbf) and change the extension to .dbf, as follows:

```
. RUN COPY Filename.crp Filename.dbf
```

2. Delete the .crp file, as follows:

```
. ERASE Filename.crp
```

3. Rename the .cpt file so that it has a .dpt extension.

Index files are only encrypted when you REINDEX or create them with an encrypted database file.

You can control when copied files are encrypted through the SET ENCRYPTION command.

## See Also

LOGOUT, SET ENCRYPTION

If you purchased dBASE IV, see Chapter 14, "dBASE Security," in *Advanced Topics* for more information about PROTECT.

If you purchased dBASE IV Developer's Edition, see Chapter 5, "dBASE Security," in *Networking with dBASE IV*.

# PUBLIC

PUBLIC designates specified memory variables and array elements that you can use and alter in any dBASE program or subprogram. Unlike private memory variables and array elements, they are not released when the program ends.

## Syntax

PUBLIC <memory variable list>/[ARRAY <array element list>]

## Usage

PUBLIC memory variables or array elements are global variables that you can use in any program. Memory variables that you want to be global must be declared PUBLIC before they are given values. Specifying a memory variable from the dot prompt declares it to be PUBLIC. System memory variables are automatically declared public. The values of system variables may be changed by any program that uses them, but the original values are returned when you quit the program.

The value of a PUBLIC memory variable may be hidden temporarily when a program or procedure file is called, by declaring memory variables with the same name PRIVATE, or by passing a parameter with the DO <procedure> WITH <parameter list> command.

Variables saved in a memory file are always RESTOREd as PUBLIC variables, if they are restored at the dot prompt. Variables restored from a memory file in a dBASE program are RESTOREd as PRIVATE variables unless you specify otherwise. To RESTORE PUBLIC variables in a dBASE program file, you first need to redeclare the variables you intend to be PUBLIC as PUBLIC variables. RESTORE FROM <filename> ADDITIVE then restores the variables as PUBLIC, overwriting any variables of the same name.

**NOTE**
*Variables you declare PUBLIC are stored as logical type variables until you initialize them.*

## Example

Within a dBASE program file, the Page and Answer memory variables are declared PUBLIC and initialized when the following commands are executed:

```
*Main.prg
DO Init_var
RETURN
PROCEDURE Init_var
PUBLIC Page, Answer
Page = 1
Answer = "Y"
RETURN
```

## See Also

DECLARE, DO, PARAMETERS, PRIVATE, RELEASE, RESTORE, SAVE, STORE

# QUIT

QUIT closes all open files, terminates the dBASE IV session, and returns control to the operating system.

## Syntax

```
QUIT
```

## Usage

This is the only safe method for exiting from dBASE IV and returning to your computer's operating system. Exiting dBASE IV by resetting your computer may damage open files and cause data loss.

# READ

READ activates all @...GETs issued since the last CLEAR, CLEAR ALL, CLEAR GETS, or READ command. It is most commonly used in dBASE program files for full-screen entry or editing data.

## Syntax

READ [SAVE]

## Usage

This command allows you to receive user input into memory variables and fields. Use it with @...GET to place the input into memory variables. When you want to receive several pieces of information instead of a one-time user input (such as with ACCEPT, INPUT, or WAIT), use several @...GET statements followed by a single READ.

The READ command uses the same full-screen editing keys as the APPEND and EDIT commands. You must use READ to edit memory variables with the @ command.

READ clears all GETs after you finish editing, unless you use the SAVE option.

## Option

READ SAVE does not clear GETs. This means that the next time you issue READ, the same set of GETs appears for editing. If using the READ SAVE option, make sure you issue CLEAR GETS before the next set of GETs.

## Programming Note

If you want to use a multi-page format (.fmt) file in which the @...SAY...GETs continue on from 2 to 32 screen pages, include a READ wherever you want a page break. The CREATE/MODIFY SCREEN command automatically installs READ commands when it generates multiple page format files.

## Example

Use the READ command to edit the contents of a memory variable:

```
. STORE SPACE(25) TO Mname
. @ 10,10 SAY "Enter your name: " GET Mname
. READ
```

READ puts the cursor into the Mname variable to allow editing. The SPACE(25) function initializes a blank character string that is 25 characters long.

## See Also

@, CLEAR, CLEAR GETS, CLEAR MEMORY, CREATE/MODIFY SCREEN, REPLACE, SET DEVICE, SET FORMAT, STORE

# RECALL

RECALL reinstates records that are marked for deletion in the active database file.

## Syntax

RECALL [ <scope> ] [FOR <condition> ] [WHILE <condition> ]

## Default

Unless otherwise specified by the <scope> or the FOR or WHILE clauses, only the current record is RECALLed.

## Special Cases

RECALL does not reinstate records that have been removed from the database by the PACK or ZAP commands.

RECALL has no effect with SET DELETED ON. If SET DELETED is ON, you must specify which records to RECALL with RECALL RECORD <n>.

## Examples

To reinstate only the first record in the database, assuming that records 1, 5, and 10 are marked for deletion with DELETE:

```
. USE Stock
. RECALL
     1 record recalled
```

To reinstate record 10:

```
. RECALL RECORD 10
    1 record recalled
```

To reinstate record 5 and any other records that have been marked for deletion:

```
. RECALL ALL
    17 records recalled
```

## See Also

DELETE, DELETED(), PACK, SET DELETED, ZAP

# REINDEX

REINDEX rebuilds all active index (.ndx) and multiple index (.mdx) files in the current work area, including the production .mdx file.

## Syntax

REINDEX

## Usage

This command reindexes all open .ndx and .mdx files in the current work area, including the tags inside .mdx files. Whenever you REINDEX files that were created with SET UNIQUE ON or with UNIQUE included in the INDEX syntax, the rebuilt index file retains its UNIQUE status regardless of whether UNIQUE is ON or OFF.

In a network environment, the database file must be in exclusive use before you REINDEX.

## Example

```
. USE Transact
. REINDEX

Rebuilding index - C:\DBASE\TRANSACT.MDX Tag: CLIENT_ID
   100 % indexed          12 Records indexed
Rebuilding index - C:\DBASE\TRANSACT.MDX Tag: ORDER_ID
   100 % indexed          12 Records indexed
```

## See Also

INDEX, KEY(), MDX(), NDX(), PACK, SET INDEX, SET ORDER, SET UNIQUE, USE, ZAP

# RELEASE

RELEASE deletes memory variables, thus opening memory space for other use. RELEASE is also used with the appropriate keyword to remove LOADed assembly language programs, menus, popups, and windows from memory.

## Syntax

```
RELEASE <memvar list>
    or
RELEASE ALL [LIKE/EXCEPT <skeleton>]
    or
RELEASE MODULE [<module name list>]
    /MENUS [<menu name list>]
    /POPUPS [<popup name list>]
    /WINDOWS [<window name list>]
```

## Usage

In the interactive mode, RELEASE ALL deletes all memory variables except system variables. Within a program, RELEASE ALL deletes only PRIVATE memory variables created in the current program or in lower level programs. It does not delete PRIVATE variables created in higher-level programs, except when used from the dot prompt.

## Options

<memvar list> is a list of names separated by commas.

RELEASE MODULE removes a LOADed module from memory. This allows you to remove LOADed assembly language program modules from memory.

RELEASE MENUS erases the listed menus from the screen and clears them from memory. All ON SELECTION and ON PAD commands associated with the menus are also cleared. A menu can't be released if it is currently in use, but it can be released once it is deactivated. If you don't specify a list of menus, all menus are released.

RELEASE POPUPS erases the named pop-up menus from the screen and from memory. It deactivates the active pop-up menu, if you specify it in the pop-up name list. Any ON SELECTION POPUP commands associated with the pop-up menus in the name list are cleared before the pop-up menus are erased. If you use no pop-up menu name list, this command erases all pop-up menus from the screen and from memory.

RELEASE WINDOWS erases specified windows from the screen and releases them from memory. The window name list contains the names of previously defined windows, separated by commas. Any text covered by the released windows is restored to the screen. You can selectively remove windows from memory with this command.

## Examples

To RELEASE all memory variables starting with the letter *m*:

```
. RESTORE FROM Mem
. RELEASE ALL LIKE m*
```

To RELEASE all memory variables except those that have the letter *x* as the third character:

```
. RESTORE FROM Mem
. RELEASE ALL EXCEPT ??x*
```

To RELEASE the Mvar, Pages, Pgx1, and Pgx2 variables:

```
. RESTORE FROM Mem
. RELEASE Mvar, Pages, Pgx1, Pgx2
```

To RELEASE a set of popups used by a menu in a financial program:

```
. RELEASE POPUPS Credit, Aging, Recvbles
```

## See Also

CALL, CALL(), CLEAR ALL, CLEAR MEMORY, LOAD, RESTORE, RETURN, SAVE, STORE

# RENAME

RENAME changes the name of a file.

## Syntax

RENAME <old filename> TO <new filename>

## Defaults

Both the old and new filenames must include file extensions. If the files are not on the default drive, precede the filenames with the path or drive designation.

## Usage

Use this command to change the names of disk files without exiting to the operating system. You can not RENAME an open file, and the new filename cannot be an existing filename in the same directory.

If you rename a database file with an associated memo file, you must also rename the corresponding memo (.dbt) file to match.

## Special Case

Do not use the letters A through J, or M, as database filenames because they are reserved for alias names. However, you can specify AA as a valid database filename.

## Examples

To RENAME a database file named School.dbf to Roster.dbf:

```
. RENAME School.dbf TO Roster.dbf
```

To RENAME the file Letters.jim to Memos.old, assuming that B is not the default drive:

```
. RENAME B:Letters.jim TO B:Memos.old
```

## See Also

CLOSE, COPY, COPY FILE, USE

# REPLACE

REPLACE changes the contents of specified fields in the active database file.

## Syntax

REPLACE <field> WITH <exp> [ADDITIVE] [,<field> WITH <exp>
    [ADDITIVE]] [<scope>] [FOR <condition>]
    [WHILE <condition>]

## Default

Unless otherwise specified by the scope or the FOR or WHILE clause, only the current record is replaced.

## Usage

REPLACE overwrites a specified field with new data. The field you select can be any type, including a memo field; however, the field and the WITH expression must have the same data type. In numeric fields, the WITH expression may be larger than the field width, in which case the number is displayed in scientific notation. In converting memo fields to character fields, REPLACE truncates the data to fit into the assigned field width.

The ADDITIVE clause lets you build a memo field from several character strings. This clause is ignored unless the field is a memo field.

**NOTE**
*Replacements on a key field of an indexed database file also update the index file if it's in use. When the replacement is made, the record moves to a new position in the index file. If you specify a scope, FOR, or WHILE when making replacements on an indexed field, all of the records you intended may not be replaced. For example, if you REPLACE ALL, only the first record and those that follow the new key field value will be replaced. That's because each record is reindexed as the REPLACE is done, and the record pointer is moved to follow the replaced record. To REPLACE all the records, you should use the database file without an index or SET ORDER TO 0 during the replace.*

## Record Pointer

The REPLACE command does not reposition the record pointer unless you specify a scope, FOR, or WHILE. If the record pointer is at the end of the file (EOF() is .T.), no replacements are made unless you specify a scope or FOR condition in the command.

## Examples

These examples use the Stock.dbf database file. The first example changes the cost for those records that match the specified condition:

```
. USE Stock
. REPLACE FOR Part_id = "C-222-1000" Item_cost WITH 1300.00
       2 Records Replaced
```

To raise the cost of all items by ten percent:

```
. REPLACE ALL Item_cost WITH Item_cost * 1.1
       17 Records Replaced
```

## See Also

ORDER, REINDEX, SET INDEX, SET ORDER, STORE

# REPORT FORM

REPORT FORM prints information from the active database or view using a report form file created by CREATE/MODIFY REPORT. You may direct a report to the screen, the printer, or to a file.

## Syntax

```
REPORT FORM <report form filename> /? [PLAIN] [HEADING <expC>]
    [NOEJECT] [SUMMARY] [<scope>] [FOR <condition>]
    [WHILE <condition>] [TO PRINTER/TO FILE <filename>]
```

## Defaults

The default extension is .frm for the FORM filename.

Unless otherwise specified by the scope, a FOR condition, or an active filter condition or query file, all records in the database are included in the report.

## Usage

First, create report form files by issuing CREATE/MODIFY REPORT. If the report form file defines records into groups, the active database or view must be INDEXed or SORTed on the same fields as the expression that defines those groups.

If the report form is in dBASE III PLUS format, the REPORT FORM command translates the file into dBASE IV format, writes the new file on disk with an .frm extension, generates a file with an .frg extension, compiles the file to an object file with an .fro extension, and runs the report from the .fro file. The old dBASE III PLUS file is renamed on disk with an .fr3 extension.

If you supply the REPORT FORM command with a filename and include an .frg or .fro extension, dBASE IV runs the report from the .frg or .fro file. If you supply the REPORT FORM command with a file extension other than .frg or .fro, or do not provide any extension at all, dBASE IV first looks for an .frg or .fro file with the same name and runs it. If it cannot find a corresponding .frg or .fro file, it assumes the file you supplied is a design file created with the CREATE/MODIFY REPORT command, and the REPORT FORM command generates an .frg file from the design file before running the report.

In multi-user operations, the REPORT FORM command places an automatic lock on the database file before printing. If another user has used RLOCK() or FLOCK() on the file, REPORT FORM generates the **File is in use by another** error message. You may copy the file to a temporary file to generate the report.

## Options

Use the query clause, ?, to activate a menu of all report forms for the active database file.

PLAIN causes the report to print without headers or footers on all except the first page of a report.

HEADING, followed by a character expression, defines an extra heading that is printed on the first line of each page. If the heading is a character string, you must delimit it. PLAIN and HEADING are mutually exclusive, with PLAIN taking precedence.

NOEJECT, with TO PRINTER, suppresses the usual initial form feed, causing the report to print on the first page that comes up in the printer.

If you set **Wrap semicolons** ON when creating the report, any semicolon (;) characters in character or memo fields will cause a carriage return and line feed when the report is run. This semicolon does not print.

TO FILE sends the report to a text (.txt) file on the default drive and displays it on the screen, unless otherwise directed.

SUMMARY suppresses display of detail lines, showing only subtotals and totals on the report.

## See Also

CREATE/MODIFY REPORT, LABEL FORM

# RESET

The RESET command removes the integrity tag from a file.

## Syntax

RESET [IN <alias> ]

## Usage

The integrity tag marks files involved in a transaction begun by a BEGIN TRANSACTION command. This tag stays on a file until the transaction is completed, or a successful ROLLBACK has occurred.

You may need to remove this tag with the RESET IN <alias> command, if you do not want to ROLLBACK a database file, or if a successful ROLLBACK is not possible.

<alias> is the alias name of a database file open in another work area. The IN clause allows you to manipulate a database file in another work area without SELECTing it as the current work area.

After issuing a RESET command, you can access the file for another transaction.

RESET should not be used in a program. Use this command at the dot prompt, when necessary, to correct an unusual situation.

## Options

The <alias> you use may be:

- A number from 1 through 10
- A letter from A through J
- An alias name, either default or supplied through the ALIAS option of the USE command
- A numeric expression that yields a number from 1 through 10 (surrounded by parentheses if it is a simple variable)
- A character expression that yields a letter from A through J or an alias name

## See Also

BEGIN TRANSACTION, END TRANSACTION, ISMARKED(), ROLLBACK, ROLLBACK()

# RESTORE

RESTORE retrieves and activates memory variables and arrays from a memory file.

## Syntax

RESTORE FROM <filename> [ADDITIVE]

## Defaults

A .mem file extension is assumed, unless you specify otherwise.

## Usage

When you RESTORE memory variables, all of the currently active variables are deleted unless you have included the ADDITIVE option. These variables are restored as PRIVATE in a dBASE program file, unless you first declare them as PUBLIC and specify ADDITIVE. Memory variables you restore from the dot prompt, however, always come back as PUBLIC.

You can have up to 25,000 memory variables available to you if you specify this amount in the Config.db file. The default number is 500. It is possible, however, that your available memory will limit the maximum number of memory variables you can use. See Chapter 6, "Customizing dBASE IV," for more information on memory variable default sizes and how you can change them in Config.db.

## Options

To retain the current memory variables, include the ADDITIVE option, which causes the RESTOREd memory variables to be added to the current ones. Any current memory variables with the same name are overwritten. To restore variables in a memory file as PUBLIC, you must redeclare them as PUBLIC before you RESTORE the memory file, and you must use the ADDITIVE option.

## See Also

PRIVATE, PUBLIC, RELEASE, SAVE, STORE

# RESTORE MACROS

RESTORE MACROS restores macros that were saved to the current macro library.

## Syntax

RESTORE MACROS FROM <macro file>

## Usage

A macro (.key) file is created when you issue the SAVE MACROS command, and contains all the macros that were in memory. RESTORE MACROS restores macros from an existing file for use during the current dBASE IV session.

If you save your macros to a file with an extension other than .key, remember to specify that extension when restoring the macro file later.

When you restore macros from a file into memory, current macros assigned to the same keys as the restored macros will be overwritten.

## Example

All macros are restored to memory with the command:

```
. RESTORE MACROS FROM Macdemo
```

This command restores all macros saved in the Macdemo.key file to memory.

## See Also

PLAY MACRO, SAVE MACROS

# RESTORE WINDOW

The RESTORE WINDOW command restores window definitions from a disk file to memory.

## Syntax

RESTORE WINDOW <window name list> /ALL FROM <filename>

## Usage

Use this command to restore window definitions from a disk file created with the SAVE WINDOW command. The default extension for window definition files is .win.

You can restore windows selectively; not all the window names have to be restored from a given .win file. You can also restore windows in any order, regardless of the order you used with the SAVE WINDOW command. If you specify ALL, then all window definitions saved in the .win file are restored.

If you used an extension other than .win when saving the window definitions, you must specify it with the filename you give the RESTORE WINDOW command.

If you restore a window name that is also currently defined in memory, the restored window definition overwrites the definition in memory.

## See Also

DEFINE WINDOW, SAVE WINDOW

# RESUME

RESUME works in conjunction with SUSPEND. It causes a SUSPENDed program to continue executing.

## Syntax

RESUME

## Usage

When you issue the RESUME command, the previously SUSPENDed command file restarts execution at the command line immediately following the line on which you stopped the file. If you entered the ROLLBACK command during a SUSPENDed transaction, the RESUME command will resume program execution at the line immediately after the END TRANSACTION command in the program.

RESUME also works with a suspended DEBUG command.

## Tip

Issuing the CLEAR command before you issue RESUME is good practice. This ensures that the commands you entered while the program was SUSPENDed do not interfere with subsequent screen output.

## See Also

BEGIN TRANSACTION, CANCEL, CLEAR, DEBUG, RETURN, ROLLBACK, SUSPEND

# RETRY

RETRY re-executes a command sequence that caused an error.

## Syntax

RETRY

## Usage

Use RETRY primarily in error recovery, since it can repeat a command until it is successfully executed. Note, however, that RETRY will not re-evaluate an IF or DO WHILE condition.

RETRY retries the command that *called the error program.* You use it with ON ERROR DO <program> to determine where the error is in your program. When RETRY is used in a program file, it closes the file. RETRY also clears out the value returned by the ERROR() function.

RETRY can be used in procedures as well as in dBASE IV program files.

## Examples

The following two command files illustrate a possible way to recover from the **File is already open** message (error number 3):

```
* Main.prg
ON ERROR DO Recover WITH ERROR()
USE My_file
RETURN
* EOF:Main.prg
```

```
* Recover.prg
PARAMETERS Error
IF Error=3
   CLOSE DATABASES
ENDIF
RETRY
RETURN
* EOF:Recover.prg
```

## See Also

ERROR(), ON ERROR, RETURN

# RETURN

Use RETURN in programs to restore control to calling programs, to the Control Center, or to the dot prompt. When control is restored to a program, the command following the calling command is executed. Alternately, RETURN is used at the end of a user-defined function to return the result of the function.

## Syntax

RETURN [ <expression> / TO MASTER/TO <procedure name> ]

## Usage

RETURN is used in a procedure file to close it and return control to the calling procedure or function, or to the dot prompt. When you include the TO MASTER option in a program, control reverts to the highest-level calling program. Specifying a <procedure name> instead of TO MASTER returns control to a particular procedure that is currently active. The MASTER option takes precedence over any procedure that may be named "master."

If you do not include the RETURN command in a procedure or function, dBASE IV puts one at the end of the procedure.

RETURN releases all PRIVATE memory variables defined in that procedure, but does not affect PUBLIC variables. RETURN also clears out any value returned by the ERROR() function. Previous values of PRIVATE system variables are restored.

RETURN is placed at the end of a user-defined function to return the result of the function. Use the <expression> statement in user-defined functions to return the function value. If the procedure determines there is no result, it returns a false (.F.). RETURN has to include an <expression> if it is being compiled as a user-defined function.

## See Also

COMPILE, DO, ERROR(), FUNCTION, PARAMETER, PRIVATE, PROCEDURE, PUBLIC, RESUME, SUSPEND

# ROLLBACK

The ROLLBACK command restores the database and index files to the state they were in before the current transaction. It closes and deletes all index and database files that may have been created during the transaction.

## Syntax

ROLLBACK [ < database filename > ]

## Usage

Use this command to undo changes made to records in the database file after you have issued a BEGIN TRANSACTION command.

You can use ROLLBACK only on the database files that are currently active in a transaction. You cannot use a database filename argument while a transaction is in progress.

ROLLBACK compares the transaction log file with the active database contents, to undo the active transaction and restore the database file to what it was at the beginning of the transaction.

Once a transaction is terminated, you can use the ROLLBACK command alone, to rollback changes in all databases, or with a filename, to rollback the changes to a particular database. In either case, ROLLBACK attempts to restore databases and index files to their state prior to the last transaction.

A transaction log file contains the pre- and post-transaction contents of each record that is altered. The ROLLBACK command refers to the transaction log file to undo transactions. ROLLBACK can fail under two circumstances:

- If there are inconsistencies between a record's pre- and post-transaction contents
- If the transaction log file is not readable

At the end of a successful ROLLBACK procedure, the database file is restored to its pre-transaction status. The transaction log file is reset to the beginning, and remains open waiting for a new transaction.

## Example

This program segment sets up for an automatic ROLLBACK with the ON ERROR command, before starting a transaction that modifies salary records:

```
ON ERROR ROLLBACK
BEGIN TRANSACTION
   REPLACE ALL Salary WITH Salary * 1.1
END TRANSACTION
```

### See Also

BEGIN TRANSACTION/END TRANSACTION, COMPLETED(), ISMARKED(), RESET, ROLLBACK()

# RUN/!

RUN executes a specified DOS command, or any program which can be executed by DOS, from within dBASE IV.

## Syntax

RUN/! <DOS command>

## Usage

RUN, and its alternate syntax !, executes the specified DOS command or program. When execution is finished, control returns to dBASE IV.

RUN requires enough additional memory for the DOS Command.com file, plus the amount required for the command or program file itself. If your computer does not have sufficient memory for the RUN command, the message **Insufficient memory** appears.

The DOS file, Command.com, must be in the current directory or the path specified by the DOS COMSPEC parameter.

Use the DOS SET COMSPEC command to tell the operating system where to find Command.com (for example, SET COMSPEC = C:\Command.com). Refer to your DOS manual for more information on SET and COMSPEC.

---

**WARNING**
Some DOS commands, such as ASSIGN and PRINT, remain in memory after being RUN. You should run these commands before loading dBASE IV. If you get an error message, clear any memory-resident programs or restart your computer and try again.

---

## Example

To reset the system date from a memory variable:

```
. Today = "02/29/88"
02/29/88
. RUN DATE &Today.
. ? DATE()
02/29/88
```

# SAVE

SAVE stores all or part of the current set of memory variables and array elements to a disk file.

## Syntax

SAVE TO <filename> [ALL LIKE/EXCEPT <skeleton>]

## Defaults

The filename must include the drive designator if the file is not on the default drive. Unless otherwise specified, the file extension for any file created with this command is .mem. If you do not use the ALL LIKE/EXCEPT option, all current memory variables are saved to the disk file.

## Usage

Use SAVE TO to save memory variables that you may work with in subsequent dBASE IV transactions.

## Examples

To SAVE all memory variables to the disk file, Myfile.mem, located on disk drive B:

```
. SAVE TO B:Myfile
```

To SAVE all memory variables with names beginning with the letter *d* to the disk file, Myfile.mem, located on disk drive B:

```
. SAVE ALL LIKE d* TO B:Myfile
```

To SAVE all memory variables, except those with names containing the letter *x* as the fourth character, to the disk file, Myfile.old, on the default drive:

```
. SAVE ALL EXCEPT ???x* TO Myfile.old
```

## See Also

PARAMETERS, PRIVATE, PUBLIC, RESTORE, SET SAFETY, STORE

## SAVE MACROS

SAVE MACROS lets you save the currently defined macros from memory to a disk file. You can restore the macro file to memory any time you want to use that set of macro definitions.

### Syntax

SAVE MACROS TO <macro file>

### Usage

Use this command to create a file with the default extension .key. You may define macro keys **Alt-F1** through **Alt-F9**, and also **Alt-F10** followed by the letters A through Z. Therefore, there are 35 unique macro keys.

You may save a minimum of one and a maximum of 35 macro definitions to each file.

Before you can overwrite an existing macro file, if SET SAFETY is ON, a message alerts you to the existence of a file with the same name.

You may specify an extension other than .key; but remember to use that special extension with the filename when restoring the file.

### Example

```
. SAVE MACROS TO Macdemo
```

This command creates a macro file called Macdemo.key, and saves the current macro definitions to it.

### See Also

PLAY MACRO, RESTORE MACROS

# SAVE WINDOW

The SAVE WINDOW command saves window definitions to a disk file.

## Syntax

SAVE WINDOW <window name list>/ALL TO <filename>

## Usage

This command creates the disk file you name, and saves the windows listed in <window name list>, or all windows in memory, to this file. Only windows that are defined and currently in memory can be saved to disk.

The SAVE WINDOW command overwrites an existing disk file if you use the same file name. SET SAFETY ON first, and you will get a prompt before the file is overwritten.

The default extension for the output disk file is .win. If you assign a different extension to the window definition file, then specify this extension with the filename to the RESTORE WINDOW command.

If you specify ALL, all the windows in memory are saved to the disk file.

## See Also

DEFINE WINDOW, RESTORE WINDOW

# SCAN

The SCAN command is a simplified alternative to the DO WHILE command. SCAN offers a more natural syntax for automatically selecting records from a data file, and for applying processing commands to those records.

## Syntax

```
SCAN [ <scope> ] [FOR <condition> ] [WHILE <condition> ]
    [ <commands> ]
    [LOOP]
    [EXIT]
ENDSCAN
```

## Usage

SCAN cycles through an active data file, acting on records that fall within the <scope>, and on records that meet the FOR and WHILE conditions. The commands you place between SCAN and ENDSCAN provide the processing for the records selected.

The <scope> option's default is every record in the file, starting from the beginning of the file. If you use NEXT or RECORD for the <scope>, scanning begins at the current record.

The FOR <condition> specifies which records within the <scope> should be acted on.

The WHILE <condition> further limits the records that may be acted on, allowing processing only as long as the WHILE conditional expression remains true (.T.).

LOOP returns control to the beginning of the SCAN process.

EXIT ends a SCAN process. Control goes to the next command following ENDSCAN.

## Record Pointer

SCAN has specific effects on the record pointer. A SCAN begins at the beginning of a file (or at the first record in the index), unless you instruct otherwise through the <scope>, or through the FOR and WHILE conditions. A SCAN continues to the end of a file (or to the last record in the index), unless the <scope>, the FOR or WHILE conditions, or the EXIT option end processing sooner. The record pointer remains where it was when processing ended.

## Examples

This simple SCAN example executes a procedure that does backorders for each uninvoiced record in the Transact data file:

```
USE Transact
SCAN FOR .NOT. Invoiced
   DO Backordr WITH Order_id, Client_id, Date_trans
ENDSCAN
```

Accomplishing the same effect as the above SCAN example with DO WHILE requires this code:

```
USE Transact
LOCATE FOR .NOT. Invoiced
DO WHILE .NOT. EOF()
   DO Backordr WITH Order_id, Client_id, Date_trans
   CONTINUE
ENDDO
```

## See Also

CONTINUE, LOCATE

# SEEK

SEEK searches for the first record in an indexed database file with a key that matches a specified expression. SEEK conducts a very rapid record search.

## Syntax

SEEK <exp>

## Usage

You normally use SEEK with memory variables. The specified expression used by SEEK can be any valid dBASE IV expression of the same data type as the index key expression.

Substring or partial key searches work only if the search expression matches the index key, starting with the character on the far left, and if SET EXACT is OFF.

For example, if the index key is *Smith, John* with SET EXACT OFF, *Smi* will find the index key. However, if SET EXACT is ON, the search expression would have to match the full length of the index key, which in this example would be *Smith, John.*

SEEK also honors any restrictions you have placed on records with the SET DELETED and SET FILTER commands.

## Record Pointer

If the search is unsuccessful, the message **Find not successful** appears, and the record pointer moves to the end of the file (EOF() is .T.). Use SET TALK OFF to suppress the message.

If you have SET NEAR ON, and your SEEK is not successful, the record pointer is positioned to the record whose index key immediately follows that of the sought-for key.

The FOUND() function is set to true (.T.) when the SEEK is successful and the record pointer is set to the found record.

## Example

The Transact database file is used to demonstrate this command.

To SEEK a character <expression>:

```
. USE Transact ORDER Order_id
. SEEK "87-115"
. ? Client_id, Invoiced, Total_bill
C00001 .F.      165.00
```

## See Also

CONTINUE, EOF(), FIND, FOUND(), INDEX, KEY(), LOCATE, MDX(), NDX(), SET DELETED, SET EXACT, SET FILTER, SET INDEX, SET NEAR, SET ORDER, TAG(), USE

# SELECT

Use SELECT to choose a work area in which to open a database file, or to specify a work area in which a database file is already open.

## Syntax

SELECT < work area name/alias >

## Usage

When you enter dBASE IV, the active work area is work area 1. The valid work areas are 1 through 10, or A through J. You can open a database file in each work area you select. Work area 10 is reserved for an open catalog. You cannot open a database file in this area if a catalog is open. If you have a database file open in work area 10 and then open a catalog, your database file will be closed automatically. Anytime you open a file in a work area containing an open database file, the first file is closed.

Each work area is also used for opening other files associated with the database file; for example, index, query, and format files. However, the same associated files cannot be open in more than one work area at the same time.

You can use SELECT to change work areas so that the database file open in the selected work area becomes the active database file. You can change the current work area by specifying the alias of the database file open in another work area or the work area number or letter. If you do not specify an alias when you open the file, the filename (without the extension) is the default alias name. The letters A through J, based on the current work area, are used as the default alias name if the filename is not a valid alias.

In dBASE IV you can still use the macro (&) function to SELECT a work area or its alias by a variable, as in dBASE III PLUS. However, dBASE IV offers a much faster and more simple method. To SELECT a work area named as a variable, place the variable in parentheses, which turns it into an explicit expression. For example, if N = 1, then SELECT (N). You can also state it without parentheses if used in an expression, as in SELECT N + 0.

Normally, commands display or change the contents of fields in the active database file. However, you can specify fields in other work areas by using an alias. dBASE IV has several commands and functions for specifying other work areas.

If you have issued the SET FIELDS command, and SET FIELDS is ON, you can also display or change data for the fields in the specified FIELDS list in other work areas.

### Record Pointer

Each work area maintains a separate record pointer. Moving between work areas does not affect the position of any record pointer.

### Examples

In this example, the Customer database file is opened in work area 1, and the Transact database file is opened in work area 3:

```
. SELECT 1
. USE Customer ALIAS Names
. SELECT 3
. USE Transact
```

To change the work area, SELECT the alias of the file or the number or letter of the work area:

```
. SELECT Names
```

In these examples, work areas were selected by using a work area number and the alias name of a database file open in a work area. You could also select a work area by a letter (for example, selecting work area 1 by entering SELECT A).

### See Also

ALIAS(), CLOSE, CONTINUE, LOCATE, SET CATALOG, SET FIELDS, SET VIEW, USE

# SHOW MENU

SHOW MENU displays a bar menu without activating it.

## Syntax

SHOW MENU <menu name> [PAD <pad name>]

## Usage

This command displays on the screen, over any existing display, the menu named in the command.

The PAD option highlights the named prompt pad when the menu appears.

If you display a bar menu with the SHOW MENU command, you cannot move the cursor in it or make selections from it. Use this command in program development to check the screen appearance of a menu you are designing.

## See Also

ACTIVATE MENU, CLEAR, CLEAR MENU, DEFINE MENU, DEFINE PAD

# SHOW POPUP

The SHOW POPUP command displays a pop-up menu without activating it.

## Syntax

SHOW POPUP <popup name>

## Usage

Use this command to display a pop-up menu without activating it. This feature is useful for checking the appearance of a popup during program development.

Because the pop-up menu is not active, the cursor cannot be moved around in it and messages associated with the pop-up menu are not displayed. The cursor remains at the dot prompt.

## Example

To display the View_pop pop-up menu:

```
. SHOW POPUP View_pop
```

The View_pop menu is displayed. Use the CLEAR command to erase the pop-up menu from the screen.

## See Also

ACTIVATE POPUP, CLEAR, DEFINE BAR, DEFINE POPUP, POPUP()

# SKIP

SKIP moves the record pointer forward or backward in a database file.

## Syntax

SKIP [ < expN > ] [IN < alias > ]

## Usage

If the database file is INDEXed, SKIP follows the index record order. If the file is not indexed, the record pointer moves sequentially by record number. SKIP also honors the current SET FILTER command, if used.

The record pointer moves forward through the records of the database file, from the current record to the end of the file, unless the result of the numeric expression is negative. A negative expression moves the record pointer from the current record toward the beginning of the file. If the expression is not included, dBASE IV moves forward one record.

If you specify IN < alias > , SKIP moves the record pointer of the work area designated by the alias name. The IN clause allows you to manipulate the record pointer in another work area without SELECTing it as the current work area. The record pointer in the original work area is not affected. If SET TALK is ON, the name of the database file precedes the record number in the display.

## Record Pointer

If you issue SKIP when the record pointer is on the last record in a file, the value of RECNO() is one greater than the number of records in the file, and EOF() is true (.T.).

If you issue SKIP −1 when the record pointer is on the first record, RECNO() remains at 1; BOF(), however, is true, which indicates that the record pointer is at the beginning of the file.

## Special Cases

Because SKIP follows the index order if an index is active, you should be especially careful when SKIPping through records of an indexed database file in a network environment. If one user edits a field that is part of the index key, or appends records to the database file, all records in the file will be reordered after the change is saved. An attempt to SKIP to or past a newly appended or changed record returns different results than if no change or addition has been made.

If another user APPENDs a record, for example, and you attempt to position on that new record with a SKIP, the record pointer will indicate the end of file and EOF() will be true. If another user edits a key field in a record, and you attempt to position past that record, the record pointer will indicate a different record than if the key field has not been changed.

To compensate for these changes in a network environment, you may issue a GO RECNO() command to reposition the record pointer on the current record, and then issue the SKIP command.

## Examples

The database files Client and Transact are used in these examples.

To move forward one record in the database file:

```
. USE Client
. SKIP
CLIENT: Record No        2
```

To move forward five records, using a numeric memory variable:

```
. Mskip = 5
              5
. SKIP Mskip
CLIENT: Record No        7
```

To move the record pointer in the Transact database file while Client is the currently selected file:

```
. USE Transact IN 2
. SKIP 4 IN Transact
TRANSACT: Record No      5
```

## See Also

BOF(), EOF(), GO/GOTO, RECNO()

# SORT

SORT creates a new database file in which the records of the active database file are positioned in the ASCII order of the specified key fields.

## Syntax

```
SORT TO <filename> ON <field1> [/A] [/C] [/D]
    [,<field2> [/A] [/C] [/D] ...] [ASCENDING]/[DESCENDING]
    [<scope>] [FOR <condition>] [WHILE <condition>]
```

## Defaults

SORTs are in ascending order (/A) unless you state /D or DESCENDING. Ascending order is the same as ASCII order (refer to Appendix G).

SORT updates a catalog if one is in use.

## Usage

You may not SORT logical or memo fields.

You can SORT on a maximum of ten fields.

A file cannot be SORTed to itself or to any other open file.

## Options

/C causes the SORT not to differentiate between upper and lower case. Therefore, /C creates a file that is not in strict ASCII order.

/A and /D apply only to one field. You may combine /C with either /A or /D. When using two options, include only one slash (for example, /DC).

ASCENDING and DESCENDING affect all fields that do not have an /A or /D option. /A or /D overrides the ASCENDING or DESCENDING options, but only for the field names preceding the /A or /D symbol.

## Tips

SORT and INDEX both reorder the records in a database file. SORT creates a new physical database file that is written on disk; INDEX creates an index (.ndx) file or multiple index (.mdx) file tag that contains pointers to each record in the original file, but does not physically rearrange the original database file. You may think of a SORT operation as analogous to an INDEX command followed by a COPY.

These commands:

```
. INDEX ON Lastname TO Lname
. COPY TO Namefile
```

perform an operation similar to

```
. SORT ON Lastname TO Namefile
```

Both SORT and INDEX arrange records in ASCII order.

The SORT command works only with a simple field or list of fields, while the index command will accept any expression (including simple fields). If you need to sort on an expression (which SORT can't do), you can INDEX on the desired expression, then use COPY to create a new sorted file.

Based on your application, you may decide that physically rearranging the data with a SORT is preferable at some times, and that CREATEing an index with pointers back to the original data records, without re-writing the database file to disk, is preferable at others. Two additional factors may influence your decision:

- The FIND and SEEK commands work only with INDEXed files. If you plan to search the database file for records or fields that match a specified value, use INDEX.
- INDEX and SORT each provide a different set of options. For example, you can INDEX a file with the UNIQUE option, so that only the first record of several records with the same key value are entered into the index. You can also index on an expression, including the substring of a character field. On the other hand, the SORT command allows you to use the FOR and WHILE clauses to extract a subset of the original database file. SORT also allows you to specify that only one field should be in reverse order, while INDEX requires that the entire expression, including all fields, be either ascending or descending. Check the options for INDEX and SORT to see which command is more useful in your application.

Do not use the letters A through J, or the letter M, as database filenames, because they are reserved for default alias names. AA, however, is a valid database filename.

When SORTing on multiple fields, place the most important key first. Separate the field names with commas.

## Example

To create a new database file called Byclient from the Transact database, sorted on Client_id in reverse chronological order:

```
. USE Transact

. SORT ON Client_id/A, Date_trans/D TO Byclient
  100% Sorted            12 Records sorted

. USE Byclient

. LIST

BYCLIENT: Record No        1
Record#  CLIENT_ID  ORDER_ID  DATE_TRANS  INVOICED  TOTAL_BILL
      1  A00005     87-113    03/24/87    .T.           125.00
      2  A10025     87-116    04/10/87    .F.          1500.00
      3  A10025     87-105    02/03/87    .T.          1850.00
      4  B12000     87-114    03/30/87    .F.           450.00
      5  C00001     87-115    04/01/87    .F.           165.00
      6  C00001     87-108    02/23/87    .T.          1250.00
      7  C00001     87-106    02/10/87    .T.          1200.00
      8  C00002     87-110    03/09/87    .T.           175.00
      9  C00002     87-107    02/12/87    .T.          1250.00
     10  L00001     87-112    03/20/87    .T.           700.00
     11  L00001     87-109    03/09/87    .T.           415.00
     12  L00002     87-111    03/11/87    .F.          1000.00
```

## See Also

COPY, INDEX

# STORE

STORE initializes one or more memory variables after creating them, if necessary, and initializes array elements previously created with the DECLARE command.

## Syntax

STORE <expression> TO <memvar list>/<array element list>

An alternative syntax for STORE is:

<memvar>/<array element> = <expression>

## Defaults

The number of memory variables you can have is the product of the MVBLKSIZE and MVMAXBLKS settings, which are contained in the Config.db file. If MVBLKSIZE = 50 and MVMAXBLKS = 10 (the default settings), you can STORE up to 500 memory variables.

Each array that you DECLARE takes only one memory variable *slot*. Dynamic memory is allocated for each array to hold the elements. Therefore, the number of array elements do not take away from the number of slots initialized by MVBLKSIZE and MVMAXBLKS.

## Usage

The data type of the variable or element is determined by the expression STOREd. For example:

```
. STORE 365.25 TO N_year
```

creates a fixed point numeric variable called N_year, if N_year did not already exist. All numeric variables created from constants, such as 365.25, are in Binary Coded Decimal format, unless the FLOAT() function converts the value to floating point binary.

An expression may be stored to several memory variables or array elements with a single STORE command. You may mix elements and memory variables in the same STORE statement, such as:

```
. STORE 1 to M_n1, M_n2, one[1,1]
```

With the alternate syntax, <memvar>/<array element> = <expression>, you may store an expression only to one memory variable or one array element at a time. The expression must be on the right side of the equal sign, and the memory variable name cannot be the same as a dBASE IV command.

If a memory variable or array of the same name already exists, it is overwritten, unless one variable or array is declared PUBLIC and the other PRIVATE.

Memory variable and array element names may contain letters, numbers, and underscores. The name may be up to 10 characters long, but must not begin with a number or underscore. Single-character names are permitted, but you should not use the single letters A through J, or the letter M, to name variables because these may conflict with alias names. Double letter combinations, such as AA, are acceptable.

Each act of storing data to a memory variable sets the data type of that variable. The type is always the same as the type of data being stored. Memory variables do not have fixed data types and they don't persist in a data type after initializing.

## Tips

If a memory variable returns an unexpected value, there may be a field with the same name in an active database file.

A field with the same name as a memory variable takes precedence in all operations except with macro substitution (&). To explicitly declare that the memory variable takes precedence, use *M->* (the symbol used to specify a memory variable) before the variable name, such as:

M-> <memvar>

You may choose to begin variable names with a distinguishing letter or letters, such as *M_*, so that they will not be confused with field names. This will also allow you to RELEASE or SAVE sets of variables and leave others intact. For example:

```
. RELEASE ALL LIKE M_*   && This releases all variables beginning with "M_".

. SAVE TO Holdmem ALL LIKE A_*    && This creates a file called Holdmem.mem,
    and saves all variables beginning with "A_" to it.
```

## Examples

To create a logical memory variable L_paid, and initialize it to false (.F.):

```
. STORE .F. TO L_paid
.F.
```

To create a floating point binary numeric variable:

```
. F_year = FLOAT(365.25)
```

To initialize the memory variables Mcost1, Mcost2, and Mcost3 to zero:

```
. STORE 0 TO Mcost1, Mcost2, Mcost3
```

To initialize array elements Month[1,2], Month[2,2], and Month[3,2] with the character strings January, February, and March, respectively:

```
. DECLARE Month[3,2]
. Month[1,2] = "January"
January
. Month[2,2] = "February"
February
. Month[3,2] = "March"
March
```

You can create date variables several ways:

```
. X = DATE()

. Y = CTOD("02/01/89")

. Newmonth = {12/31/88}
```

## See Also

&, CLEAR ALL, CLEAR MEMORY, DECLARE, LIST/DISPLAY MEMORY, PARAMETERS, PRIVATE, PUBLIC, RELEASE, RESTORE, SAVE

# SUM

SUM totals numeric expressions to memory variables or arrays.

## Syntax

SUM [ < expN list > ] [TO < memvar list > /TO ARRAY < array name > ]
    [ < scope > ] [FOR < condition > ] [WHILE < condition > ]

## Defaults

Unless otherwise specified by a scope or a FOR or WHILE clause, all database records are SUMmed.

If you do not provide an expression list, all numeric fields are SUMmed.

## Usage

The sum for each numeric expression is placed in the memory variable list or array. The sum of the first expression is placed in the first variable of the list or first array element, and processing continues for each item in the expression list. If there are insufficient memory variables or elements in the array, only the existing variables or elements are filled. If there are more variables in the list or more array elements than needed, the extra variables or elements will not be filled and will contain their original values.

The named array must be one-dimensional. SUM produces a type F numeric variable.

## Example

To sum the Total_cost and Qty fields to the array Mitems:

```
. USE Stock
. DECLARE Mitems[2]
. SUM Item_cost, Qty TO ARRAY Mitems
17 records summed
Item_cost   Qty
    10505   20
. ? STR(Mitems[2],3,0), "pieces were sold to date."
20 pieces were sold to date.
. ? "The sum of the totals is $" + LTRIM(STR(Mitems[1],9,2))
The sum of the totals is $10505.00
```

## See Also

AVERAGE, CALCULATE, DECLARE, FIXED(), FLOAT(), STORE, TOTAL

# SUSPEND

SUSPEND is a debugging command that lets you interrupt a running program. Once the program is suspended, you can enter commands at the dot prompt. Use RESUME to resume execution or CANCEL to clear the suspended program from memory.

## Syntax

SUSPEND

## Usage

You can suspend execution by placing the SUSPEND command within a program, or by typing X at the **ACTION:** prompt in the debugger.

If SET TRAP is OFF and you press the **Esc** key while a program is running, dBASE IV prompts **Cancel Ignore Suspend**. If you press **Esc** when SET TRAP is ON, dBASE IV presents the debugger screen.

If you respond with Suspend, you can either change a dot prompt command line stored in the history buffer or execute new commands. Commands from program files are not stored in the history buffer. You cannot modify the SUSPENDed dBASE program file or other open dBASE program or procedure files. If you try, dBASE IV responds with **File already open**. You can, however, modify the file if you are executing it from within the debug screen by using the Edit option of Debug.

When you are ready to continue program execution, type RESUME.

If you create any memory variables while the file is suspended, they are PRIVATE at the level of the suspension.

CANCEL cancels all DO files, including ones that are suspended. You must close any procedure file with CLOSE PROCEDURE.

If you attempt to return to the program by typing DO <filename> at the dot prompt while the program is suspended, you may eventually run out of memory. The suspended program remains in memory, waiting for you to RESUME, CANCEL, or QUIT. Use RESUME to continue execution of the program, rather than running the program again from the beginning. Use CANCEL to close all open program files.

## See Also

CANCEL, COMPILE, DEBUG, DO, RESUME, SET ECHO, SET PROCEDURE, SET STEP, SET TALK, SET TRAP

# TEXT

TEXT outputs blocks of text to the screen or printer. This command provides a simple, convenient way to write to the output device.

## Syntax

TEXT
< text characters >
ENDTEXT

## Usage

The text characters are output exactly as originally entered into a program.

Text output goes to any active streaming output device such as the screen, the printer, or an alternate file.

The & (macro) function is not expanded when embedded in the text, and is output exactly as typed.

SET PRINTER ON to send text to the printer.

## Example

The following two sentences of text are displayed on the screen if this TEXT...ENDTEXT block is contained in a program:

```
TEXT
This is an example of text that is to be displayed by the
TEXT command. This text is routed directly to the screen
without being interpreted by dBASE IV.
ENDTEXT
```

## SEE ALSO

@, ?/??, ???, LIST/DISPLAY, SET PRINTER

# TOTAL

TOTAL sums the numeric fields of the active database file and creates a second database file to hold the results. The numeric fields in the TO database file contain the totals for all records that have the same key value in the original database.

## Syntax

```
TOTAL ON <key field> TO <filename> [FIELDS <fields list>]
    [<scope>] [FOR <condition>] [WHILE <condition>]
```

## Defaults

The TO filename must include the drive designator if it is not on the default drive. dBASE IV assumes a .dbf extension unless you specify otherwise.

Unless otherwise specified by the scope, or a FOR or WHILE clause, all records are TOTALed. All numeric fields in the record are totaled unless limited by the FIELDS list.

If a catalog is in use, the TO file is added to it.

## Usage

The active database file must be either INDEXED or SORTed on the key field. If the TO filename exists, TOTAL gives a warning prompt before overwriting the file, unless SET SAFETY is OFF.

All records with the same key field become a single record in the TO database. All numeric fields appearing in the FIELDS list contain totals. All other fields contain the data from the first record of the set of records with identical keys.

The structure of the TO file is the same as the active database file's, except that memo fields are not copied to the new file. Type N numeric fields in the source file produce type N numeric totals in the TO file, and type F fields produce type F totals.

## Tips

Do not use the single letters A through J, or the letter M, as database filenames, because they are reserved for default alias names. AA, however, is a valid database filename.

If a field size in the TO file is not large enough to accommodate the total, dBASE IV places asterisks in the field (indicating an overflow). To avoid this, use MODIFY STRUCTURE to increase the size of the fields in the active database file to a size that can handle the largest total.

## Example

To create a new database file called Amounts from the Transact database file, with the sum of Total_bill for each Client_id:

```
. USE Transact ORDER Client_id
Master index: CLIENT_ID

. TOTAL ON Client_id TO Amounts
  12 records totaled
   7 records generated

. USE Amounts
. LIST Client_id, Total_bill

Record#  CLIENT_ID  TOTAL_BILL
      1  A00005         125.00
      2  A10025        3350.00
      3  B12000         450.00
      4  C00001        2615.00
      5  C00002        1425.00
      6  L00001        1115.00
      7  L00002        1000.00
```

## See Also

AVERAGE, CALCULATE, INDEX, MODIFY STRUCTURE, SET SAFETY, SORT, SUM

# TYPE

TYPE displays the contents of an ASCII text file.

## Syntax

TYPE <filename> [TO PRINTER/TO FILE <filename> ] [NUMBER]

## Defaults

The filename must include the file extension. If the file is not on the default drive, and SET PATH is not set to the file's location, you must precede the filename with its drive and directory location.

Files with extended ASCII characters are displayed as graphic characters.

## Usage

TYPE displays standard ASCII text files only, such as command or procedure files. TYPEd database files, index files, and database text files cannot be easily read because they contain other information that dBASE IV uses. TYPE cannot specify an open file; in other words, a program may not include a command to TYPE itself or any other open file.

If your printer driver doesn't support graphics characters, such as double lines or corners, ordinary printable characters such as hyphens and plus signs are substituted.

## Options

If you include the NUMBER keyword in the command, line numbers will be displayed or printed. Line numbers correspond to the physical lines in the program, not lines in the display. Line continuations in the program count as separate lines. A page heading consisting of the filename, the date, and the page number is also displayed unless SET HEADING is OFF. The complete filename as entered on the command line is displayed on the left, followed by the current date in the SET DATE format. Page numbers are displayed on the right of the page.

If you include TO PRINTER, the output will be sent to the printer.

## See Also

SET DATE, SET HEADING, SET PATH, SET PRINTER

# UNLOCK

UNLOCK releases record and file locks so that other users can modify data.

## Syntax

UNLOCK [ALL/IN <alias>]

## Usage

Once locked, a record or file can't be modified or updated by another user until the UNLOCK command has been executed to unlock the record or file.

The UNLOCK command unlocks the records or the file in the selected work area that were locked by the current user. UNLOCK ALL releases all locks, both file and record locks, in all work areas. UNLOCK IN <alias> unlocks the file lock or record locks in the specified work area. The ALL and IN keywords are mutually exclusive.

This command releases all record locks, if a record lock was the last lock thrown. It releases the file lock, if a file lock was the last lock thrown. Certain commands automatically lock the file or record before execution. These also release the lock after execution, and you usually do not need to issue an UNLOCK unless the command did not complete its execution.

If you lock a file or record that is related to other open files or records, all the files or records in the relation will be locked. Unlocking the file or record automatically unlocks all related files or records.

## Example

The following example illustrates using the RLOCK() function to lock a record and the UNLOCK command to unlock it:

```
. USE Client
. ? RLOCK()
.T.
. REPLACE Lastname WITH "Nelson"
     1 record replaced
. REPLACE Firstname WITH "John"
     1 record replaced
. UNLOCK
```

## See Also

FLOCK(), RLOCK()/LOCK(), SET RELATION

# UPDATE

UPDATE uses data from another database file to replace fields in the current database file. It makes the changes by matching records in the two files based on a single key field.

## Syntax

```
UPDATE ON <key field> FROM <alias>
    REPLACE <field name 1> WITH <expression 1>
    [,<field name 2> WITH <expression 2>...]
    [RANDOM]
```

## Usage

The database file being UPDATEd must be the active file. The FROM database file is an open file in another work area, specified by its alias.

The key field must have the same name in both database files. If there isn't a unique key field entry in the UPDATE file for each record being received from the FROM file, then only the first record with a matching key field value is updated. In the case of multiple matching records in the FROM file, the matching record in the UPDATE file will receive all of these updates and retain only the last values received.

## Options

Both files must be ascendingly sorted or indexed on the key field, unless RANDOM is used. RANDOM requires the UPDATEd database to be indexed on the key field; the FROM file, however, may be in any order. You should not specify the key field in the REPLACE option, although it is allowed. Key fields need to be kept intact for matching purposes.

If the REPLACE expression involves a field in the FROM database file (which is usually the case), the WITH field must be identified as alias->field.

## Example

This section of an example program updates the Total_bill field of the Transact database file with the product of the Qty and Item_cost fields of the Stock database file:

```
USE Transact ORDER Order_id
REPLACE ALL Total_bill WITH 0
USE Stock ORDER Order_id IN 2
UPDATE ON Order_id FROM Stock REPLACE Total_bill WITH Stock->Qty *;
      Stock->Item_cost
LIST
```

The program would then display this result:

```
Record#  CLIENT_ID  ORDER_ID  DATE_TRANS  INVOICED  TOTAL_BILL
      1  A10025     87-105    02/03/87    .T.           650.00
      2  C00001     87-106    02/10/87    .T.          1200.00
      3  C00002     87-107    02/12/87    .T.          1250.00
      4  C00001     87-108    02/23/87    .T.          1250.00
      5  L00001     87-109    03/09/87    .T.           165.00
      6  C00002     87-110    03/09/87    .T.           100.00
      7  L00002     87-111    03/11/87    .F.          1000.00
      8  L00001     87-112    03/20/87    .T.           700.00
      9  A00005     87-113    03/24/87    .T.           125.00
     10  B12000     87-114    03/30/87    .F.           450.00
     11  C00001     87-115    04/01/87    .F.           165.00
     12  A10025     87-116    04/10/87    .F.          1000.00
```

## See Also

INDEX, JOIN, REPLACE, SORT, TOTAL

# USE

USE opens an existing database file, and may open .mdx and .ndx index files. If the database contains memo fields, the associated .dbt file is opened automatically.

## Syntax

```
USE [<database filename>/?] [IN <work area number>]
    [[INDEX <.ndx or .mdx file list>]
    [ORDER [TAG] <.ndx filename>/<.mdx tag> [OF
     <.mdx filename>]] [ALIAS <alias>] [EXCLUSIVE] [NOUPDATE]]
```

## Defaults

Unless IN <work area number> is included, dBASE IV opens the database file in the currently selected work area.

USE ↵ closes the database and any associated index files in the currently selected work area. USE IN <work area number> ↵ closes the files in the named work area.

Unless you specify otherwise, dBASE IV assumes the .dbf extension for the database filename and the .mdx or .ndx extension for the indexes specified in an index list. If you specify a filename in the index list without specifying a file extension, dBASE IV first attempts to open an .mdx file. If it cannot find an .mdx file, dBASE IV attempts to open an .ndx file.

If the database filename isn't already recorded, USE <database filename> updates a catalog when one is open and SET CATALOG is ON.

## Usage

USE ? displays a menu of cataloged .dbf files when a catalog is open, or of all .dbf files in the current directory when a catalog is not open.

The IN <work area number> clause can be used to open files in another work area. The numeric expression may range from 1 to 10 or from A to J. You can also use numeric expressions that yield 1 to 10 or character expressions that yield A to J.

Indexes determine the order in which records from the database file are presented. Index files with an .ndx extension are compatible with earlier versions of dBASE III and dBASE III PLUS. They contain a set of pointers to the database file which, when the index is invoked, present the records in *index order*. Index files with an .mdx extension may contain up to 47 tags. Each tag in the .mdx file is an index and acts as an .ndx index. When you open an .mdx file, you use only one DOS file handle, but you can access up to 47 indexes. When you open an .ndx file, you use one DOS file handle, and you have access to only one index.

All open indexes, including .mdx tags, are updated whenever a change is made to the associated database file. Up to ten .ndx files, plus the production .mdx file, may be opened for each database file.

Each .ndx and .mdx file uses a DOS file handle. Open files are limited to 99, or the number specified by the FILES setting in CONFIG.SYS, and also by available memory. Five file handles are reserved by dBASE IV, leaving a possible maximum of 94.

After opening the database file with USE, you can open and close all index files and tags with SET INDEX. You can also close .ndx files with CLOSE INDEX and DELETE TAG.

## Options

The ORDER clause determines which index controls the index order. The index may be either an index (.ndx) file or a tag contained within a multiple index (.mdx) file; either of these may be specified. The controlling index contains the index key expression that the FIND and SEEK commands use for their search. If only one index is specified, that index controls the index order and the ORDER clause is not needed.

You can indicate the .mdx file that contains the tag by using the clause OF <.mdx filename>. If a tag is not in the production .mdx file, OF should be used to specify the correct index.

The SET ORDER command switches the controlling index without closing and re-opening .ndx and .mdx files.

ALIAS allows you to provide an alias name that can be used later to refer to the database file. If no alias name is included, the alias is the same as the database filename. You can also use the letters A through J, or the numbers 1 through 10, to refer to the ten work areas. When you select a new work area, the database file in that work area becomes the active database file.

Alias names may contain only letters, numbers, or underscores (_), and they must start with a letter. Filenames don't have this restriction, so there are cases where the filename cannot be used as an alias. When the file name cannot be used as an alias, the letter of the current work area (A to J) is used.

EXCLUSIVE opens the database file with an exclusive file open attribute in a networking environment. The file is not shared, and other users cannot USE the database file until you close it with a CLEAR ALL, CLOSE, or USE command. In a single-user environment, all files are opened EXCLUSIVE, and dBASE IV ignores this keyword in the USE command line.

NOUPDATE prevents any additions, deletions, or changes to the data in the file being used. In effect, it makes the file read-only.

## Record Pointer

When you USE a database file without any index files, the record pointer is positioned at the first record in the file.

When you USE a database file with one or more index files, the record pointer is positioned at the first logical record of the controlling index. The record pointer follows the order of that index until another index is invoked with SET ORDER or SET INDEX.

## Special Case

Unless you provide an alias with the ALIAS keyword, dBASE IV usually assigns the filename as the default alias name. Some valid filenames, such as X-y, cannot be used as alias names because they contain characters that dBASE reserves for other uses. If a filename contains characters that prohibit it from being used as the default alias, dBASE assigns a letter alias, from A to J, as the default.

If SET TALK is ON, no ALIAS option is specified, and the filename can't be used as the default alias, the work area letter is used, and you see the message **Default alias is A** (or the appropriate work area letter).

## Example

To open the Client, Transact, and Stock database files in work areas 1, 2, and 3, respectively:

```
. CLOSE ALL

. USE Client INDEX Cus_name
Master index: CUS_NAME

. USE Transact ORDER Client_id IN 2
Master index: CLIENT_ID

. USE Stock ORDER Order_id IN 3
Master index: ORDER_ID
```

## See Also

CLEAR ALL, CLOSE, COPY INDEX, COPY TAG, DBF(), DELETE TAG, INDEX, SELECT, SET CATALOG, SET EXCLUSIVE, SET INDEX, SET ORDER, SET REPROCESS

# WAIT

WAIT causes all dBASE IV processing to pause until any key is pressed.

## Syntax

WAIT [ <prompt> ] [TO <memvar> ]

## Defaults

The memory variable will receive a null value (ASCII value 0) if you press ↵. If you enter any other nonprinting character, the memory variable will yield the graphic character of the same ASCII value.

## Usage

The prompt is a character expression. If you do not give a prompt, the message **Press any key to continue** appears.

## Option

If you include the TO <memvar> option, a character type memory variable is created which stores the keyboard entry.

## Example

You can use WAIT to temporarily halt execution of a dBASE program. To do this, you could include the following commands in your program:

```
WAIT "Do you wish to continue? (Y/N) " TO M_con
IF UPPER(M_con) = "N"
   RETURN
ENDIF
```

## See Also

ACCEPT, INPUT, ON KEY, STORE

# ZAP

ZAP removes all records from the active database file.

## Syntax

ZAP

## Usage

ZAP is equivalent to, but faster than, a DELETE ALL command followed by PACK.

If SET SAFETY is ON, dBASE IV responds **Zap <filename> ? (Y/N)**.

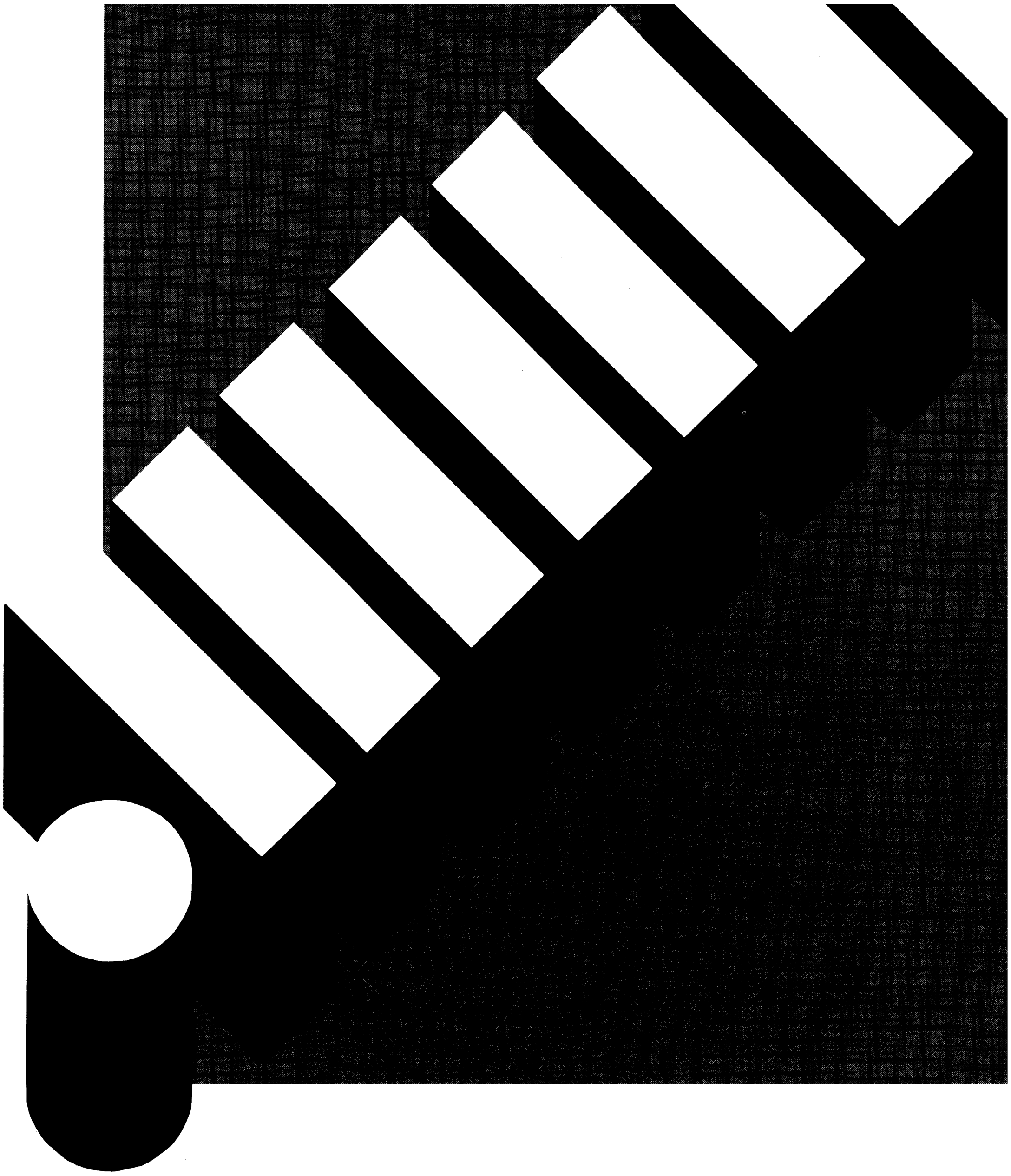
Any open index files in the current work area are automatically reindexed to reflect the empty database file.

dBASE IV reclaims the space previously used by the ZAPped file, including that of any associated memo or index files.

## See Also

DELETE, PACK, SET SAFETY

# SET Commands



**i 1 2 3 4 5 6 A B C**

**D E F G In**

# SET Commands

## SET

SET is a full-screen, menu-driven command for displaying and changing the values of many SET commands.

### Syntax

SET

### Usage

At the dot prompt, type SET to display the **SET** menu. The menu bar at the top of the screen displays five selections.

The **Options** submenu shows the default settings for many SET commands. You may change the settings from this menu.

The **Display** submenu changes the color and intensity display attributes associated with text, headers, the status line and fields. You can see the actual screen appearance of your selections in a side window as you make them. When you are satisfied with the appearance of your screen, press **Ctrl-End**. This mode of exit retains any changes made to a SET command for the duration of your current session in dBASE IV. When you quit dBASE IV, all values except display revert to the program defaults. **Esc** lets you leave without saving your selections.

The **Keys** submenu lets you program certain function keys and key combinations.

The **Disk** submenu lets you change the default disk drive and the drive search path.

The **Files** submenu lets you create alternate, format, device, or index file types.

## See Also

For a complete explanation of the SET menu screen, please refer to *Using the Menu System.*

You can also change the default settings from the Config.db file. Refer to Chapter 6, "Customizing dBASE IV," for more information on editing the Config.db file.

# SET ALTERNATE

SET ALTERNATE records all output other than that of full-screen commands to a text file.

## Syntax

SET ALTERNATE on/OFF

SET ALTERNATE TO [ < filename > [ADDITIVE]]

## Default

The default for SET ALTERNATE is OFF.

The filename must include the drive designator if the file is not on the default drive. The file extension .txt is assumed unless otherwise specified.

SET ALTERNATE TO ↵ closes an open ALTERNATE file. CLOSE ALTERNATE is an alternative syntax.

## Usage

This command consists of two parts:

SET ALTERNATE TO < filename > creates a text file and opens the file. It overwrites any file with the same name.

The ADDITIVE option positions the cursor to the end of the alternate file before any output is written to it, so that an existing file can be appended to without being overwritten.

SET ALTERNATE ON starts the recording of keyboard entries and screen displays in the alternate file. Full-screen operations such as @...SAY, EDIT, BROWSE, and APPEND are not recorded.

You can use SET ALTERNATE OFF to suspend writing to the text file without closing the text file. You can then start recording again with SET ALTERNATE ON. The text file will not be closed until you issue the CLOSE ALTERNATE or SET ALTERNATE TO command. Be sure you issue the CLOSE ALTERNATE command before using the text file. The file created by SET ALTERNATE is a standard ASCII text file which you can edit with MODIFY COMMAND or a word processor.

If you do not want a blank line at the beginning of the alternate file, change the ? to ??. The command ? always issues a carriage return, line feed before printing text.

## Examples

The commands below illustrate the SET ALTERNATE commands. To create the alternate text file, type:

```
. SET ALTERNATE TO Textfile
```

Commands executed after you open an alternate file are not recorded until you SET ALTERNATE ON. Type:

```
. SET ALTERNATE ON
```

All dBASE commands executed are stored in the alternate file until you close the alternate file or suspend recording of commands.

To suspend recording of commands, type:

```
. SET ALTERNATE OFF
```

To resume writing to the text file, type:

```
. SET ALTERNATE ON
```

To close the text file, type:

```
. CLOSE ALTERNATE
```

To demonstrate the ADDITIVE option, create a file called Alt_out.txt using the MODIFY FILE command. Enter the following text:

**Mary had a little lamb.**

Store this file. Now execute the following commands in a program file:

```
SET ALTERNATE TO Alt_out.txt ADDITIVE
SET ALTERNATE ON
? "whose fleece was white as snow."
```

The file Alt_out.txt now contains:

```
Mary had a little lamb
whose fleece was white as snow.
```

## See Also

CLOSE

# SET AUTOSAVE

The SET AUTOSAVE command saves each record to the disk after a single I/O operation.

## Syntax

SET AUTOSAVE on/OFF

## Default

The default for SET AUTOSAVE is OFF.

## Usage

The SET AUTOSAVE command reduces chances of data loss by saving records to the disk after each change to a record. The default is OFF to allow you to abandon edits or changes you are not sure of. When SET AUTOSAVE is OFF, dBASE saves records to disk as the record buffer is filled.

You can turn SET AUTOSAVE ON from the dot prompt, or select ON from the SET menu, or make it a part of your custom configuration in the Config.db file.

## See Also

Chapter 6, "Customizing dBASE IV"

# SET BELL

SET BELL controls the audible tone which alerts you to the end of a field, or to any erroneous or invalid entry. In addition to turning it off or on, you may also set its frequency and duration.

## Syntax

SET BELL ON/off

SET BELL TO [ < frequency > , < duration > ]

## Defaults

The default for SET BELL is ON. The default frequency is set to 512 hertz, and the duration is set to 2 ticks. Each tick is approximately 0.0549 seconds.

## Usage

The available frequency range is between 19 to 10,000 cycles per second. To get a lower pitched sound, use frequencies between 20 to 550. To get a higher pitched sound, use frequencies between 550 to 5,500. The duration can be from 2 to 19 ticks. The changes take effect immediately, and last for the duration of the dBASE IV session. To use your own bell settings each time you start dBASE IV, enter your BELL settings in the Config.db file as explained in Chapter 6, "Customizing dBASE IV."

To restore the default, type:

```
. SET BELL TO ↵
```

## Examples

To test the bell frequency and duration from the dot prompt, type:

```
. SET BELL TO 20,1

. ?CHR(7)
```

or:

```
. SET BELL TO 500,12
. ?CHR(7)
```

# SET BLOCKSIZE

SET BLOCKSIZE changes the default blocksize of memo fields and multiple index (.mdx) files.

## Syntax

SET BLOCKSIZE TO <expN>

## Usage

You may use a numeric argument from 1 to 32. This command changes the default blocksize of memo files by multiplying the argument by 512 to get the actual blocksize in bytes. The default blocksize is 1, which is the only size that is compatible with dBASE III PLUS.

After the blocksize has been changed, memo fields that are created with CREATE/MODIFY STRUCTURE and COPY will have the new blocksize.

If an existing memo file has an inefficient blocksize, use the SET BLOCKSIZE command to change it. Then copy the database file to a new file. The new file will have the new blocksize.

## Tips

Large blocks tend to provide faster string manipulation, but may slow down I/O processing. Small blocks may provide slower string manipulation, but better performance.

If your memo fields tend to be large, you may increase the blocksize to 7 or 8. Too large a blocksize may slow processing speed, and increase the .dbt file size.

You may also change the BLOCKSIZE parameter in the Config.db file. See Chapter 6, "Customizing dBASE IV."

## See Also

COPY, CREATE/MODIFY STRUCTURE

# SET BORDER

The SET BORDER command changes the default border of menus, windows, and @ commands from a single-line box to several user-defined border characters.

## Syntax

SET BORDER TO [SINGLE/DOUBLE/PANEL/NONE/
    < border definition string > ]

The border definition string may be defined as:

    [ < 1 > ] [,[ < 2 > ] [,[ < 3 > ] [,[ < 4 > ] [,[ < 5 > ]
    [,[ < 6 > ] [,[ < 7 > ] [,[ < 8 > ]]]]]]]

where each number 1 through 8 is a decimal value for any IBM ASCII character. The required order for specifying the sides and the corners using 1 through 8 is illustrated in Figure 3-1.

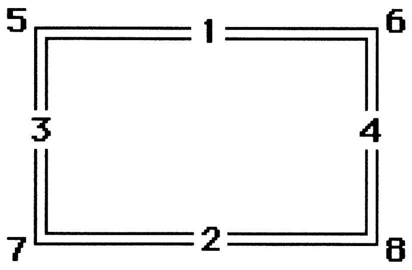


Figure 3-1 Order of border characters

## Default

The default border is a single line box.

## Usage

You can change the default border with the SET BORDER command to a double line box, to an inverse video panel, to any other ASCII character, or to no border at all.

SET BORDER TO DOUBLE changes the border to a double line box.

SET BORDER TO ↵ restores the default single line box.

SET BORDER TO PANEL gives an inverse video box as if ASCII 219 were entered.

The border setting also affects @...SAY and @...TO commands and window borders. Lines drawn with the @ commands use the SET BORDER default.

## Examples

To set a border that is double lines on top and bottom, single lines at the sides, and single line corners, enter the command:

```
. SET BORDER TO DOUBLE
. SET BORDER TO ,,179,179,213,184,212,190
```

First you set the border to double lines. Then you define a new border, using the ASCII values for the line-drawing characters of the IBM extended character set. The two commas retain double lines for the top and bottom borders. The value 179 sets the left and the right side to a single line. The corner characters combine horizontal double lines and vertical single lines. Thus, they match the top and bottom borders. The two SET BORDER TO commands can be combined into one command as follows:

```
. SET BORDER TO 205,205,179,179,213,184,212,190
```

To set a border that is an inverse video bar, enter the command:

```
. SET BORDER TO PANEL
```

## See Also

@...SAY, @...TO

# SET CARRY

SET CARRY updates all fields, or only specified fields, depending on the syntax you use.

## Syntax

SET CARRY on/OFF

This command carries forward all changes made in APPEND and INSERT.

SET CARRY TO [ < field list > [ADDITIVE]]

This command brings the specified fields from the previous record to the new record only when you change them using APPEND, BROWSE, or INSERT.

## Default

The default for SET CARRY is OFF.

The command does not affect INSERT BLANK or APPEND BLANK; a blank record is still added.

## Usage

When SET CARRY is ON, all fields are brought forward into the next record.

If you use SET CARRY TO < field list > , only the specified fields are updated. This command ignores SET FIELDS, allowing you to carry fields forward in BROWSE window.

If you use the ADDITIVE option, the fields in the fields list are added to the list of previous fields specified with SET CARRY.

SET CARRY TO with no fields list returns to the default condition where all the fields are brought forward.

If SET CARRY is OFF, no fields are updated. Issuing a SET CARRY TO automatically turns SET CARRY to ON so that the fields list can be carried forward.

## See Also

APPEND, BROWSE, EDIT, INSERT, READ, SET FIELDS, SET FORMAT

# SET CATALOG

The SET CATALOG command is used to create or open a catalog file. When a catalog is open and SET CATALOG is ON, new files which you use or create are added to the catalog.

## Syntax

SET CATALOG on/OFF

SET CATALOG TO [<filename>/?]

## Default

The default for SET CATALOG is OFF, and no catalogs are selected. However, selecting a catalog with SET CATALOG TO automatically SETs CATALOG ON.

## Usage

Use SET CATALOG TO <filename> to open an existing catalog or, if one doesn't exist, to create a new catalog. Catalog filenames are limited to eight characters. dBASE IV always assigns the extension (.cat) to catalog filenames.

dBASE IV searches for a catalog first in the dBASE IV home directory, and if a catalog is not found, it looks next in the current drive and directory. If you SET TITLE ON when you create a new catalog, dBASE IV asks you to enter a one-line description of the catalog file. The description you enter for each catalog entry is displayed later when you use the SET CATALOG TO ? command to select and highlight a catalog name.

A master catalog file, Catalog.cat, is created the first time you create a catalog. dBASE IV always checks to see if this main catalog exists, and creates it if it does not exist. The master catalog is stored in the dBASE IV home directory by default and keeps track of other catalogs. You may delete it or move it to another directory. If deleted from the home directory, the master catalog is stored on the current directory. You can create different master catalogs in other drives or directories.

The master catalog allows you to use the SET CATALOG TO ? command. It stores catalog filenames along with the catalog title descriptions.

Catalogs are database files which have a predefined file structure. When you create or select a catalog file, it is automatically opened in work area 10. When a catalog is open, work area 10 is reserved for the system to update the catalog file, so you cannot select and use that work area for another database file.

When you open or create a new catalog, SET CATALOG is automatically set ON. From then on, all database files (and associated files such as index, query, format, report, and label files) you use, or new files you create, are added to the catalog. With SET CATALOG ON, the open catalog records entries for files associated with open database files. The catalog is updated automatically by certain commands such as CREATE, INDEX, and CREATE/MODIFY REPORT or CREATE/MODIFY SCREEN.

Whenever you open a catalog, dBASE IV checks the catalog contents against the disk. If you've previously deleted any files with SET CATALOG OFF, dBASE IV updates the catalog by deleting the files that are in the catalog but have been deleted from the disk.

If you decide you want to close a catalog, use the command SET CATALOG TO with no argument. If you just want to stop adding files to an open catalog, you can SET CATALOG OFF. Then, when you want to resume adding files to the catalog, SET CATALOG ON again.

SET CATALOG OFF, unlike turning off the catalog with SET CATALOG TO with no argument, keeps the catalog open and allows you to use the query clause (?) with other dBASE commands. For example, with SET CATALOG OFF, you can type MODIFY REPORT ?, and dBASE IV activates a menu list of .frm files in the open catalog. Because the files in the catalog are linked with the database file they were created or used with, only the files relevant to the active database file are listed.

**NOTE**
*If SET CATALOG is OFF when a database file is used and is later set ON, files you use with the open database file (for example, an index file) are not added to the catalog. dBASE IV displays the message **File not cataloged, since SET CATALOG was OFF when the active database file was USEd**.*

## Catalog Structure

All catalogs have the same file structure. The structure is shown in Table 3-1.

Table 3-1 Catalog structure

| Field | Field Name | Type | Width | Dec |
|---|---|---|---|---|
| 1 | Path | Character | 70 | |
| 2 | File_name | Character | 12 | |
| 3 | Alias | Character | 8 | |
| 4 | Type | Character | 3 | |
| 5 | Title | Character | 80 | |
| 6 | Code | Numeric | 3 | |
| 7 | Tag | Character | 4 | |
| ** Total ** | | | 181 | |

With the exception of the Title and Tag fields, dBASE IV automatically fills in the field contents whenever a new file is entered in the catalog.

### Path

Enter the path statement when the catalog is not on the default drive and path.

### File_name

This is the name of the file, including its extension.

### Alias

If you do not assign an alias name, dBASE IV uses the database filename as the alias name. This field is left blank for all other file types.

### Type

This field contains the default file extension that identifies the type of file. Even if you specified a different extension when creating the file, the default extension is entered in the Type field. However, the extension that you specified is included in the File_name field.

### Title

This is an optional field. If SET TITLE is ON, dBASE IV prompts you to add a description when you create the catalog. You have up to 80 spaces to describe the contents of the new catalog. If you want to change this description, and the catalog is not in USE, type:

```
. USE <catalog name.cat>
. EDIT
```

If the catalog *is* in USE:

```
. SELECT 10
. EDIT
```

### Code

When you have a catalog open, Code is the number dBASE IV assigns to each database file put into USE.

Program files are assigned 0. A database file is assigned a number when it is created. Each new database file gets the next higher code number. Files associated with a database file such as index, format, label, query, report form, screen, and view are assigned the same code number as the database file they reference.

### Tag

This field is not used.

## Adding Entries

When SET CATALOG is ON, a new entry is added automatically to the active catalog when you use any of the following commands:

| | |
|---|---|
| COPY STRUCTURE | IMPORT FROM |
| COPY STRUCTURE EXTENDED | INDEX |
| CREATE | JOIN |
| CREATE FROM | SET FILTER TO FILE |
| CREATE/MODIFY LABEL | SET FORMAT |
| CREATE/MODIFY QUERY | SET VIEW |
| CREATE/MODIFY REPORT | SORT |
| CREATE/MODIFY SCREEN | TOTAL |
| CREATE/MODIFY VIEW | USE |

If the filename already exists in the catalog, you are prompted for a file title (if SET TITLE is ON). Each of these commands also allows you to select files using the ? query from the currently open catalog, regardless of whether SET CATALOG is ON or OFF.

**NOTE**
*Use the SET TITLE OFF command to suppress the file title prompt, if you do not want to enter file titles. To prevent the catalog from being updated, use the SET CATALOG OFF command.*

## See Also

SET TITLE

# SET CENTURY

SET CENTURY allows the input and display of century prefixes in the year portion of dates.

## Syntax

SET CENTURY on/OFF

## Default

The default for SET CENTURY is OFF. Digits representing the century are not displayed and cannot be entered.

The default date entry and display allows only two digits for the year entry (current twentieth century only). However, if a date calculation results in a non-twentieth century date, this value can be placed into a date field, and it will retain the correct century. If the field is edited with a full-screen editing command and SET CENTURY is OFF, the year is changed to a twentieth-century date.

Although the four-digit year displayed with SET CENTURY ON makes the date display ten characters, the database field size is always eight bytes. This is because dates are stored internally as numbers. Also, the format in which a date is displayed does not affect its internal storage.

## Usage

With SET CENTURY ON, the date display contains a four-digit year. When SET CENTURY is OFF, the year is displayed in two digits, with the twentieth century assumed.

SET CENTURY affects all full-screen editing commands and date displays. When SET CENTURY is OFF, you can input only twentieth century dates.

## Example

```
. ? DATE()
02/14/88
. SET CENTURY ON
. ? DATE()
02/14/1988
. SET CENTURY OFF
```

## See Also

CTOD(), DATE(), DTOC(), DTOS(), SET DATE, YEAR()

# SET CLOCK

SET CLOCK controls the display and screen position of the system clock.

## Syntax

SET CLOCK on/OFF

SET CLOCK TO [ < row > , < column > ]

## Defaults

The default setting for the clock is OFF. When the clock is turned on, the default setting is at row 0, column 69.

## Usage

Use the SET CLOCK command to display the system clock and to determine its location. SET CLOCK TO turns the clock on and positions its display.

SET CLOCK TO without parameters resets the clock coordinates to 0, 69 from any other location.

The clock display is not suppressed with full-screen menu commands, regardless of its setting.

# SET COLOR

SET COLOR allows the selection of colors and display attributes for use on color and monochrome monitors.

## Syntax

SET COLOR ON/OFF

SET COLOR TO [[ < standard > ] [,[ < enhanced > ] [,[ < perimeter > ]
    [,[ < background > ]]]]]

where standard and enhanced settings specify foreground and background colors separated by /.

SET COLOR OF NORMAL/MESSAGES/TITLES/BOX/HIGHLIGHT
    /INFORMATION/FIELDS TO [ < color attribute > ]

allows setting or changing the color attributes of individual display areas.

## Usage

SET COLOR ON/OFF switches between color and monochrome monitors on systems equipped with both. The default is whatever the system is using when you start dBASE IV.

SET COLOR TO ↵ resets the screen to dBASE IV default colors which are coded into the program. This command does not reset to the color definitions you may have included in the Config.db file.

Each time you start dBASE IV, any color settings you have placed into the Config.db file will override the program defaults.

Use the SET COLOR TO command to select the colors to display text, background, and highlighted areas on your screen.

On an enhanced graphics adapter (EGA or compatible color graphics adapters), when you select the colors from the SET command full-screen menu, you can see the selections displayed in a small side menu. You can toggle blinking on or off by pressing **B** from the **SET** menu.

The settings revert to the default colors when you quit dBASE IV. To store a color setup in the Config.db file as your default setting, refer to Chapter 6, "Customizing dBASE IV."

You can change the color settings by specifying letter codes for each of the types of displayed text: standard, enhanced, border, and background. The attribute letters used to select colors are listed in Table 3-2.

Table 3-2 Color table

| **Color** | **Attribute Letter** | **Color** | **Attribute Letter** |
|---|---|---|---|
| Black | N or blank | Red | R |
| Blue | B | Magenta | RB |
| Green | G | Brown | GR |
| Cyan | BG | Yellow | GR+ |
| Blank | X | White | W |
| Gray | N+ | | |

To specify blinking, place an asterisk (*) after the color letter. To specify blanking, place an X for the color letter. To specify high intensity, place a plus ( + ) next to the color letter (except for brown, yellow, and gray colors). The hardware of the EGA card does not allow you to set high intensity colors as the background. If you do, attributes included with the background settings are applied to foreground settings.

For display adapters that allow you to set different colors for standard and enhanced text areas, set the *standard* and *enhanced* colors as foreground/background and enhanced foreground/background pairs. For example, SET COLOR TO G/B, GR+ /N sets *standard* colors to green on a blue background, and *enhanced* colors to yellow on a black background.

For display adapters that require the same background for standard and enhanced areas, specify the foreground colors individually, and then set the background color. For example, SET COLOR TO B, G, R displays standard text as blue over red standard display areas, and enhanced (highlighted) text as green on red with a green on red border.

You can also selectively change one of the color settings. For example, when specifying foreground/background pairs, you can omit the foreground or background color. When you do not specify a color, either before or after the slash in the foreground/background pair, the color black is selected.

If you do not want characters in a specific area to appear on your screen (for instance, for a password entry), specify Blank (X) for either a foreground or background color in those areas.

For a monochrome monitor, you can specify the letters U for underline and I for inverse video instead of using colors to provide emphasis for standard and enhanced foreground text. If you do specify colors, or the letters U or I, make sure they work properly for the particular display adapter and application you are running.

The syntax of SET COLOR OF...TO controls predefined screen area groupings so that you can change color attributes of individual screen areas independent of one another.

Standard and enhanced screen area groups are summarized in Table 3-3.

Table 3-3 Color attribute groups of screen areas

| **Attribute Group Names** | **Standard — All Controlled Areas of Screen** |
|---|---|
| NORMAL | Uncontrolled @...SAY output |
| | Unselected fields in BROWSE |
| | Layout editor design surface (dim) |
| | Unselected text on design surface |
| | Unselected uncolored display only field templates |
| | Static memo, window borders |
| | Conditions in QBE file skeleton |
| | Conditions in QBE condition box |
| | Calculated field expressions |
| | Unselected, uncolored box borders in Layout editor |
| | Uncolored box borders drawn with @...TO command |
| | User-entered text in the Applications Generator |
| MESSAGES | Message line bright messages |
| | Navigation line messages |
| | Available, unselected menu and list choices (bright) |
| | Unavailable menu and list choices (dim) |
| | Error box interiors |
| | File window contents |
| | Help box interiors |
| | Bolded Help box text (bright) |
| | Prompt box interiors |
| | Unselected prompt box buttons (bright) |
| | Unselected error box buttons (bright) |
| | Unselected Help box buttons (bright) |

(*continued*)

Table 3-3 Color attribute groups of screen areas (*continued*)

| **Attribute Group Names** | **Standard — All Controlled Areas of Screen** |
|---|---|
| TITLES | List headings<br>Help box headings<br>Control Center banner<br>Control Center catalog name heading and filename<br>Control Center filename and file description headings<br>BROWSE field name headings<br>BROWSE table grids (dim)<br>Database design column labels<br>Database design field numbers<br>Directory name at top of file window<br>Files: heading<br>Sorted by: heading<br>Unselected report design band lines<br>QBE file skeleton field names and filename<br>QBE view skeleton field names and view name<br>QBE view skeleton grid (dim)<br>QBE file skeleton grids (dim)<br>QBE calculated fields skeleton grids (dim)<br>QBE condition box border (dim)<br>QBE condition box heading<br>Ruler line<br>Text styles defined with the **Style** menu (bright)<br>End-of-Report line in report design<br>Text that is underlined in Help screens (bright)<br>Static text in Applications Generator forms<br>Underlined Help box text (bright)<br>Database design grid |
| BOX | Menu borders<br>File and sorted by information<br>File window border<br>List borders<br>Prompt box borders<br>Label design layout box border<br>Unselected Applications Generator object border |

(*continued*)

Table 3-3 Color attribute groups of screen areas (*continued*)

| **Attribute Group Names** | **Enhanced — All Controlled Areas of Screen** |
|---|---|
| HIGHLIGHT | Highlighted menu and list choices<br>Highlighted prompt box buttons<br>Selected static text on design surface<br>Selected field on design surface<br>Selected box<br>Control Center filename and file description<br>Flying cursor in ruler line<br>Report design band line labels<br>Report design selected band lines<br>Information box borders<br>Information box interiors<br>Current field in the Applications Generator<br>Static text under the cursor on design surface<br>Box under the cursor on design surface<br>Field under the cursor on design surface |
| INFORMATION | Clock<br>Error box borders<br>Help box borders<br>Status line<br>Selected button in error box<br>Selected button in Help box |
| FIELDS | Prompt box data entry areas<br>Selected field in BROWSE<br>Editable fields in @...GET<br>Database design in selected field<br>Editable field design surface field templates<br>Unselected fields in the Applications Generator |

## Special Case

The older Compaq portable computers with built-in monochrome monitors, and some other systems, function as though they have color monitors. Thus, monochrome attributes U and I do not generate underlining and inverse video. The color settings are translated to shades of black and green. On these monitors, NORMAL bright is not the same color as MESSAGES bright.

## Examples

To set a color display with a standard display that uses enhanced intensity white letters on a blue background, and an enhanced display that uses yellow letters on a black background, at the dot prompt type:

```
. SET COLOR TO W+/B,GR+/N
```

Because it is easier to see menu selections against a different background, it is recommended that you use a different background color on the enhanced display.

The status line is part of the INFORMATION grouping. You can set the status line to a different color pair, for example enhanced white on red. Type:

```
. SET COLOR OF INFORMATION TO W+/R
```

and the status line changes to intense white on red while the screen remains intense white on blue for regular text, and yellow on black for fields and menu bars. When displayed, the clock has the same colors as the status line.

## See Also

SET, SET DISPLAY

# SET CONFIRM

SET CONFIRM determines cursor movement at the end of an entry field.

## Syntax

SET CONFIRM on/OFF

## Default

The default for SET CONFIRM is OFF.

## Usage

SET CONFIRM OFF causes the cursor to advance from one entry field to the next when the first entry field is filled.

SET CONFIRM ON causes the cursor to remain at the last space in an entry field until ↵ is pressed.

# SET CONSOLE

SET CONSOLE turns the screen display on and off from within a program.

## Syntax

SET CONSOLE ON/off

## Default

The default for SET CONSOLE is ON.

## Usage

SET CONSOLE works only within a program and suppresses output to the screen. Use SET CONSOLE OFF to prevent the display of reports and programs routed to the printer.

You cannot SET CONSOLE OFF from the dot prompt.

When the console is off, keyboard input is allowed. You can type in input requested by commands such as WAIT and ACCEPT; however, the screen remains blank. You can see neither the prompts from these commands nor what you are typing.

@...SAY...GETs override the CONSOLE setting and are visible regardless of the SET CONSOLE status.

SET CONSOLE OFF does not suppress error messages or safety prompts.

# SET CURRENCY

The SET CURRENCY command changes the symbol used for currency.

## Syntax

SET CURRENCY TO [ < expC > ]

## Usage

This command changes the symbol associated with the currency unit used in monetary data and calculations. dBASE IV allows a maximum of nine characters to be used for this symbol.

Usually, changing the currency symbol also involves changing the decimal point to a comma, and changing the numeric separator to a period. When used with PICTURE clauses, the SET CURRENCY command changes the currency symbol that is displayed.

## Examples

```
. SET CURRENCY TO " DM"
. STORE 123456.78 TO Number
. @ 10,0 SAY NUMBER PICTURE "@$"
```

results in:

```
DM123456.78
```

An example of using the $ PICTURE template follows. Notice that you do not enter the separator character when storing a number.

```
. SET POINT TO ","
. SET SEPARATOR TO "."
. SET CURRENCY TO "DM"
. STORE 123456.78 TO Number
. @ 10,0 SAY Number PICTURE "$$$,$$$,$$$.99"
```

results in:

```
DDDM123.456,78
```

To prevent the letter D from repeating, put a space as the first character of the currency symbol and repeat the command. The result is:

```
DM123.456,78
```

# SET CURRENCY LEFT/RIGHT

SET CURRENCY LEFT/RIGHT changes the position of the currency symbol from the left of the currency amount to the right of the currency amount.

## Syntax

SET CURRENCY LEFT/right

## Default

The default for SET CURRENCY is LEFT.

## Usage

The default is for the currency symbol to be on the left of the currency amount. This command positions the symbol to the right for currencies that use the right convention, such as the French franc. To change the default, modify the Config.db file as described in Chapter 6, "Customizing dBASE IV."

## Examples

To change the currency display to the right of the amount for the French franc, type:

```
. SET CURRENCY RIGHT
. SET CURRENCY TO "F"
. SET POINT TO ","
. SET SEPARATOR TO "."
. STORE 5000.25 TO AMOUNT
. @ 10,0 SAY AMOUNT PICTURE "@$"
5000,25F
```

# SET DATE

SET DATE determines the format for date displays.

## Syntax

SET DATE [TO] AMERICAN/ansi/british/french/german/italian/japan/usa/mdy/dmy/ymd

## Default

The default setting for DATE is AMERICAN.

## Usage

This command allows flexibility in date input and output. With it, you can quickly change the date output format.

You can also change the default date format by placing this option in the Config.db file. See Chapter 6 for customizing the Config.db file.

The date formats available to the SET DATE command are:

| | |
|---|---|
| AMERICAN | mm/dd/yy |
| ANSI | yy.mm.dd |
| BRITISH/FRENCH | dd/mm/yy |
| GERMAN | dd.mm.yy |
| ITALIAN | dd-mm-yy |
| JAPAN | yy/mm/dd |
| USA | mm-dd-yy |
| MDY | mm/dd/yy |
| DMY | dd/mm/yy |
| YMD | yy/mm/dd |

## Examples

```
. Date = CTOD({11/05/87})
11/05/87
. SET DATE ANSI
. ? Date
87.11.05
. SET DATE BRITISH
. ? Date
05/11/87
```

## See Also

DATE(), DMY(), MDY(), SET CENTURY

# SET DEBUG

SET DEBUG is a tool for locating errors in a program. It determines whether output from SET ECHO is sent to the screen or printer.

## Syntax

SET DEBUG on/OFF

## Default

The default for SET DEBUG is OFF.

## Usage

When SET DEBUG is ON, the output of SET ECHO is routed to the printer. This prevents interference between the program's screen operations and the screen displays that are normally generated by the debugging process.

With SET DEBUG OFF, the output of SET ECHO is routed to the screen.

With both SET DEBUG ON and SET ECHO ON, outputs for the SET TALK (for example, the number of records indexed), LIST and DISPLAY commands, and error messages are *not* sent to the printer unless SET PRINTER is ON.

## See Also

DEBUG, SET ECHO, SET TALK

# SET DECIMALS

SET DECIMALS determines the number of decimal places dBASE IV displays for the results of numeric functions and calculations.

## Syntax

SET DECIMALS TO <expN>

where <expN> is a numeric expression between 0 and 18.

## Default

The default is two decimal places.

## Usage

SET DECIMALS applies to division, multiplication, and all mathematical, trigonometric, and financial calculations.

The maximum number of decimals is 18. This means that a maximum of 20 numbers including the decimal point can be displayed.

## Example

Compare the results of several calculations using different decimal place settings:

```
. ? 3/4
 0.75
. ? SQRT(4.37)
2.09
. SET DECIMALS TO 4
. ? 3/4
 0.7500
. ? SQRT(4.37)
2.0905
```

# SET DEFAULT

SET DEFAULT allows the user to select the drive where all operations take place and all files are stored.

## Syntax

SET DEFAULT TO <drive> [:]

## Default

The default drive is the drive from which you started dBASE IV.

When you change drives with SET DEFAULT TO, the default directory is the directory last specified on that drive. For example, if you go to drive A from drive C subdirectory \DATA, when you return to drive C you will be in \DATA.

SET DEFAULT does not check whether or not the named drive exists and does not change the drive to which you return when you quit dBASE IV.

Refer to Chapter 6, "Customizing dBASE IV," to find out how to set the default drive in the Config.db file.

## Example

If the Customer database file is located on disk drive A but dBASE IV was started from drive C, typing the following command:

```
. USE Customer
```

results in the message **File does not exist**. To change the default drive and open the database file, type:

```
. SET DEFAULT TO A:
. USE Customer
```

## See Also

SET PATH

# SET DELETED

SET DELETED determines whether records that are marked for deletion are included or ignored by other dBASE IV commands.

## Syntax

SET DELETED on/OFF

## Default

The default for SET DELETED is OFF.

## Usage

INDEX and REINDEX always include all records regardless of the status of SET DELETED.

With SET DELETED ON, most commands do not consider deleted records part of the file. For example, dBASE IV will not LOCATE or LIST deleted records. However, if you DISPLAY a scope of records or GOTO a specific record, then you will see records marked for deletion.

If SET DELETED is ON, RECALL ALL does not recall any records, because the deleted records are ignored.

Also, SEEK, FIND, SEEK(), and deleted records are pointed to by a relation.

## See Also

DELETED(), SET FILTER

# SET DELIMITERS

SET DELIMITERS determines how field widths are indicated in the full-screen mode.

## Syntax

SET DELIMITERS on/OFF

SET DELIMITERS TO <expC>/DEFAULT

## Default

The default for SET DELIMITERS is OFF so that entry fields are not enclosed by specified delimiter characters. While SET INTENSITY is ON, entry fields are displayed in inverse video.

SET DELIMITERS ON uses colons (:) to set off the field areas, unless you choose a different delimiter with the SET DELIMITERS TO command.

## Usage

SET DELIMITERS TO <expC> defines the characters used to mark the field area. The character string must be one or two characters. If you define one character, that character marks the beginning and the end of the field. If you define two characters, the first marks the beginning of the field and the second marks the end of the field. If you define more than two characters, only the first two are read.

SET DELIMITERS must be ON for the symbols defined by SET DELIMITERS TO to be in effect.

## Examples

To mark the beginning and the end of a field with #:

```
. SET DELIMITERS TO "#"
. SET DELIMITERS ON
```

To mark the beginning of each field with [ and the end with ]:

```
. SET DELIMITERS TO "[]"
. SET DELIMITERS ON
```

To reset the delimiters to colons, either enter no parameters or the keyword DEFAULT:

```
. SET DELIMITERS TO DEFAULT
```

The @...GET command uses the characters defined by the most recently issued SET DELIMITERS command. The following dBASE IV program file provides an example:

```
SET TALK OFF
SET DELIMITERS TO "[]"
SET DELIMITERS ON
one = "abc"
two = "123"
CLEAR
@ 10,10 GET one
READ
SET DELIMITERS TO DEFAULT
@ 11,10 GET two
READ
```

## See Also

@, APPEND, CHANGE, EDIT, INSERT, READ, SET INTENSITY

## SET DESIGN

SET DESIGN prevents transfers to design mode from the dot prompt or from the Control Center. It inactivates the **Shift-F2** key combination, and prevents going to the dot prompt or QBE designer from EDIT/BROWSE modes.

### Syntax

SET DESIGN ON/off

### Default

The default is ON.

### Usage

This command is intended for use by an applications developer. Its purpose is to prevent a user from entering into design mode (that is, using Database, Report, Form, Label, Query, and Applications from the Control Center). Thus, a developer can use the Control Center menu system as the interface for a limited application and keep the user out of the functionality of dBASE IV.

# SET DEVELOPMENT

SET DEVELOPMENT activates a checking program in dBASE IV which compares the creation dates of the compiled object (.dbo) file with its program source (.prg) file to prevent you from using an outdated object file.

## Syntax

SET DEVELOPMENT ON/off

## Default

The default for SET DEVELOPMENT is ON.

## Usage

When ON, this command makes dBASE IV compare the creation date and time of the program source (.prg) file and the compiled object (.dbo) file. The dBASE IV internal editor (MODIFY COMMAND) and the Compiler delete the compiled .dbo file when its associated source file is modified. Then, dBASE IV automatically recompiles the newly modified source file, and creates a new dbo file.

If SET DEVELOPMENT is OFF, dBASE IV does not compare the dates and times of source and object files.

## See Also

COMPILE, MODIFY COMMAND

# SET DEVICE

SET DEVICE determines whether @...SAY commands are routed to the screen, the printer, or a file.

## Syntax

SET DEVICE TO SCREEN/printer/file <filename>

## Default

The default is SET DEVICE TO SCREEN.

## Usage

With SET DEVICE TO PRINTER, the output from @...SAY commands is sent to the printer. @...GET commands are ignored. An @ command that specifies coordinates of lower values than previous @ commands results in a page eject.

On some printers, @...SAYs may not appear until the print buffer is emptied. To empty the buffer, issue an EJECT command or print a blank line.

You can also send the output from an @...SAY command to a file by specifying a new file name with the keyword FILE.

## Example

To redirect @...SAYs to a file:

```
. SET DEVICE TO FILE Text.txt
. @ 1,1 SAY "This is on the first line."
. @ 5,5 SAY "This is on the fifth line."
. SET DEVICE TO SCREEN
. TYPE Text.txt
This is on the first line.




This is on the fifth line.
```

## See Also

@, EJECT, SET FORMAT

# SET DISPLAY

SET DISPLAY selects between a monochrome and a color monitor, and determines the number of lines displayed by most graphics cards that support both 25- and 43-line screens.

## Syntax

```
SET DISPLAY TO MONO/COLOR/EGA25/EGA43/MONO43
```

## Usage

You can use this command only if you have hardware that is capable of supporting monochrome and/or color display and an EGA or equivalent graphics card that supports 43-line display. If you do not have the required hardware, then this command has no effect on the display mode, and gives an error message.

## See Also

SET, SET COLOR OF, SET COLOR TO

# SET DOHISTORY

SET DOHISTORY is retained in dBASE IV for compatibility with dBASE III PLUS.

## Syntax

SET DOHISTORY on/OFF

## Default

The default for SET DOHISTORY is OFF.

## Usage

This command is not used in dBASE IV.

# SET ECHO

SET ECHO displays command lines from dBASE IV programs on the screen and/or the printer as they are executed (from the dot prompt, or from a dBASE program file).

## Syntax

SET ECHO on/OFF

## Default

The default for SET ECHO is OFF.

## Usage

Program instructions are not normally displayed during program execution. SET ECHO ON is one of four dBASE IV commands that can be used with DEBUG in debugging dBASE program execution. The other three commands are SET DEBUG, SET STEP, and SET TALK.

When both SET ECHO and SET DEBUG are on, the output of the SET ECHO command is redirected to the printer.

## See Also

DEBUG, SET DEBUG, SET PRINT, SET STEP, SET TALK

# SET ENCRYPTION

SET ENCRYPTION establishes whether a newly created database file is encrypted if PROTECT is used. Without PROTECT, SET ENCRYPTION has no effect on the encryption status of any file.

## Syntax

SET ENCRYPTION on/OFF

## Default

The default for SET ENCRYPTION is OFF.

## Usage

This command determines whether copied files (that is, files created through the COPY, JOIN, and TOTAL commands) are created as encrypted files. An encrypted file contains data enciphered into another form to hide the contents of the original file. An encrypted file can only be read after the encryption has been deciphered or copied to another file in decrypted form.

To access an encrypted file, you must enter a valid user name, password, and group name after the log-in screen prompts. Your authorization and access levels determine whether you can or cannot copy an encrypted file.

After you access the file, SET ENCRYPTION OFF to copy this file to a decrypted form. You need to do this if you wish to EXPORT the file. You must also SET ENCRYPTION OFF to use the options of the COPY TO command.

**NOTE**
*Encryption works only with PROTECT. If you do not enter dBASE IV through the log-in screen, you will not be able to use encrypted files.*

## Tips

All encrypted files used in an application must have the same group name.

Encrypted files cannot be JOINed with unencrypted files. Make both files either encrypted or unencrypted before JOINing them.

Encrypted files must be copied to new files in decrypted form before they can be used with the COPY STRUCTURE EXTENDED or MODIFY STRUCTURE commands. After you copy the file you may modify its structure. You can encrypt the new file with PROTECT by assigning it an access level.

You can encrypt files created through the CREATE command by assigning the files an access level through PROTECT.

## See Also

PROTECT

# SET ESCAPE

SET ESCAPE lets you halt the processing of a dBASE IV program by pressing the **Esc** key. It lets you stop screen scrolling by pressing ←, or **Ctrl**-S. It also allows you to record these three keys with INKEY().

## Syntax

SET ESCAPE ON/off

## Default

The default for SET ESCAPE is ON.

## Usage

With SET ESCAPE ON, if you press **Esc** while a command is being processed or a dBASE IV program is running, the processing stops and dBASE IV displays an error box:

***** INTERRUPTED *****
**Cancel, Ignore, Suspend? (C, I, or S)**

With SET ESCAPE OFF, the **Esc** key is disabled and command processing cannot be interrupted. SET ESCAPE OFF also disables the operation of **Ctrl**-S or ← when these keys are used to temporarily suspend scrolling.

**NOTE**
*When SET ESCAPE is OFF, you cannot terminate program execution without rebooting your computer. Use SET ESCAPE OFF only in tested programs that do not get in endless loops. Rebooting your computer to interrupt program execution may cause data loss.*

## See Also

INKEY(), ON KEY, ON ERROR, READKEY()

# SET EXACT

SET EXACT determines whether a comparison between two character strings requires the strings to be the same length.

## Syntax

SET EXACT on/OFF

## Default

The default for SET EXACT is OFF.

## Usage

If SET EXACT is OFF, comparisons between character strings begin with the left character in each string and continue character-by-character to the end of the string on the right of the relational operator. If the two strings compare favorably up to that point, the result of the comparison is true (.T.).

If SET EXACT is ON, the comparison of characters in each string is the same, except that both character strings must be the same length for the comparison to be evaluated as true (.T.).

## Examples

dBASE IV optimizes constants during compilation. Comparing two constant string expressions will produce different results if SET EXACT is ON or OFF. The expression:

```
" abcd " = " abcde "
```

is a logical true (.T.) if SET EXACT is OFF and a logical false (.F.) if SET EXACT is ON.

Expressions that are optimized during compilation are not re-evaluated during program execution. If you SET EXACT OFF, dBASE IV optimizes and stores the above expression as a logical true (.T.). If you next SET EXACT ON during program execution, this expression still returns a logical true (.T.).

# SET EXCLUSIVE

SET EXCLUSIVE allows a user to open a database file on a multi-user system for exclusive use, so that no other user may have *any* access to that file.

## Syntax

SET EXCLUSIVE on/OFF

## Default

The default for SET EXCLUSIVE is OFF.

## Usage

It is not possible for any other user to read or write to a file which has been opened for exclusive use.

Both the CREATE and the SAVE commands SET EXCLUSIVE ON whenever you use them.

## See Also

COPY INDEXES/TAG, INDEX ON, PROTECT, SET ENCRYPTION

# SET FIELDS

SET FIELDS defines a list of fields that may be accessed in one or more files. It also determines whether that list is used as the default fields or expression list for dBASE commands that utilize a fields list. It sets read-only flags, and supports calculated fields and wildcards to match field names.

## Syntax

```
SET FIELDS on/OFF

SET FIELDS TO [ <field> [/R] / <calculated field id> ...]
              [, <field> [/R] / <calculated field id> ...]

SET FIELDS TO ALL [LIKE/EXCEPT <skeleton> ]
```

## Default

The default for SET FIELDS is OFF. It is set ON when you specify a fields list with SET FIELDS TO. Do not use SET FIELDS ON without first specifying a fields list.

## Usage

The field options can include a list of database field names and calculated field identifiers. The /R provides an optional read-only flag for database field names only.

The calculated field identifier name can equal any valid dBASE expression. For example:

```
Gross_Pay = Salary * Hours
```

where Gross Pay is a calculated field that is set equal to the expression Salary * Hours.

The skeleton uses the wildcard character * for any number of characters and ? for one character. ALL LIKE selects fields that match the skeleton. ALL EXCEPT selects the fields that do not match the skeleton.

When SET FIELDS is OFF (the default), all the fields of the active database are available. Fields in files open in other work areas are available for display only if you specify them prefixed with their alias names.

SET FIELDS TO defines a list of fields that may be accessed in one or more files. You can write to database files in other areas by specifying an alias. It also redefines the default field or expression list used with dBASE IV commands which use or display fields. The list of fields specified by SET FIELDS TO is not active unless SET FIELDS is ON.

SET FIELDS TO ↵ removes those fields from the list that belong to the active database file. SET FIELDS TO ALL includes all the fields of the active database file.

After a fields list is set, additional or consecutive SET FIELDS TO commands add fields to the current fields list. To include fields from database files in other work areas, SELECT the work area of the file and use the SET FIELDS TO command, or specify the database file's alias with the field name.

When SET FIELDS is ON, the LIST STRUCTURE and DISPLAY STRUCTURE commands indicate the fields which are in the currently defined fields list with the > symbol.

**NOTE**
*SET FIELDS TO can define a list of fields from multiple files, but it does not relate those files. You must use SET RELATION to establish a connection between files in different work areas.*

## Affected Commands

The fields list that you define with the SET FIELDS command is used by all commands that have a field list option in their syntax. The following commands use the fields list specified by SET FIELDS TO:

APPEND
AVERAGE
BROWSE
CALCULATE
CHANGE
CREATE/MODIFY VIEW FROM ENVIRONMENT
COPY TO
COPY STRUCTURE
DISPLAY
EDIT
EXPORT
JOIN
LIST
SUM
TOTAL

The SET FIELDS TO command does not check to see if files are related; you must verify this yourself.

---

**WARNING**
To ensure data integrity when using view files, or the SET FIELDS and SET RELATION commands, you should add records to one database file at a time, with all fields of the database file accessible.

Changes to key information should be done knowing that if you change *parent* key information without also changing corresponding key information in related files, you will lose the relation between those records. Do not construct field lists from multiple unrelated files unless you have a sound reason for doing so.

---

You may use indexes even if the index expression contains fields that are not in the current fields list.

LOCATE, SET FILTER, and SET RELATION also ignore the fields list; these commands can access all fields in all open database files. With all other commands and functions, you can access only those fields contained in the SET FIELDS list.

If you use multiple database files, an alias may be used to distinguish between fields that have the same name in different work areas. For example, you can use A-> and B-> to differentiate between identical field names in different work areas.

When referring to fields without the use of an alias, dBASE IV resolves ambiguities according to these guidelines:

- If SET FIELDS is OFF, only the active database file is searched.
- If SET FIELDS is ON, the fields list is searched for the first match with the field name you specify.

## Examples

Display the Part_name, Item_cost, and Qty from the Stock database file. Create a calculated field called Total to display the product of Item_cost and Qty.

```
. Use Stock
. SET FIELDS TO Part_name, Item_cost, Qty, Total="$"+STR(Item_cost*Qty,8,2)
. GOTO 10
. LIST Stock: Record 5

Record#    Part_name     Item_cost   Qty      Total
     10    CHAIR, DESK     1000.00     1      $ 1000.00
     11    CHAIR, SIDE      350.00     2      $  700.00
     12    BOOK CASE        125.00     1      $  125.00
     13    LAMP, FLOOR      150.00     3      $  450.00
     14    LAMP, FLOOR      165.00     1      $  165.00
```

Use the ALL LIKE option of SET FIELDS TO on the Client database file to access only those files that have the string "name" in the field name.

```
. SET FIELDS TO ALL LIKE *.name.
. Use Client
. LIST

Record#      LASTNAME      FIRSTNAME
      1      Wright        Fred
      2      Bailey        Sandra
      3      Martinez      Ric
      4      Peters        Kimberly
      5      Yamada        George J.
      6      Timmons       Gene
      7      Beluga        Yuri
      8      Beckman       Riener
```

## See Also

CLEAR FIELDS, CREATE/MODIFY VIEW, CREATE VIEW FROM ENVIRONMENT, SET RELATION, SET SKIP, SET VIEW

# SET FILTER

SET FILTER allows display of only those records of a database file that meet a specified condition.

## Syntax

SET FILTER TO [FILE <filename> /?] [ <condition> ]

## Default

SET FILTER TO ↵ turns off the filter for the active database file.

SET FILTER TO FILE adds a filter (query) file to a catalog if one is open and SET CATALOG is ON.

## Usage

SET FILTER applies only to the database file open in the work area where the command is issued. Therefore, you can set a different filter condition for each open database file.

All commands that require a database file to be in USE, such as AVERAGE, BROWSE, EDIT, and REPORT, can use conditions specified by SET FILTER.

SET FILTER TO FILE <filename> reads the filter condition from a file created by CREATE/MODIFY QUERY. If a catalog is open, you can use the catalog query clause to display a menu of all query files for the active database file. dBASE IV will accept a dBASE III PLUS query file; however, this file cannot be modified using dBASE IV.

SET FILTER TO <condition> sets up a filter based on a valid dBASE IV expression. The condition can filter records in the active database file based on any of the allowed data types; for example, on a character field, SET FILTER TO Lastname = "Jones"; or on a date field, SET FILTER TO Departure = {01/01/88}. Both conditions can be present together.

Filters aren't activated until after the record pointer is moved within a file. The best way to activate a filter setting is to execute a GO TOP or SKIP command to move the record pointer.

## Examples

To filter the Transact database file for all records whose Date_trans field is on or after 3/20/87:

```
. Use Transact
. SET FILTER TO Date_trans >={03/20/87}
. LIST

Record#  Client_id  Order_id  Date_trans  Invoiced  Total_bill
      8  L00001     87-112    03/20/87    .T.           700.00
      9  A00005     87-113    03/24/87    .T.           125.00
     10  B12000     87-114    03/30/87    .F.           450.00
     11  C00001     87-115    04/01/87    .F.           165.00
     12  A10025     87-116    04/10/87    .F.          1500.00
```

Specific record pointer positioning ignores the SET FILTER TO conditon. Continuing with the environment of the preceding example:

```
. GOTO 3
Transact: Record No 3
. DISPLAY

Record#  Client_id  Order_id  Date_trans  Invoiced  Total_bill
      3  C00002       87-107  02/12/87    .T.          1250.00
```

## See Also

CREATE/MODIFY QUERY, SET DELETED

# SET FIXED

SET FIXED is retained in dBASE IV for compatibility with dBASE III PLUS; it has no effect on the number of decimals displayed.

## Syntax

SET FIXED on/OFF

## Default

The default for SET FIXED is OFF.

## Usage

This command is not used in dBASE IV. In dBASE IV the SET DECIMALS command determines the number of decimal places displayed in calculation results.

## See Also

SET DECIMALS

# SET FORMAT

SET FORMAT allows a form to be used with the READ, EDIT, APPEND, INSERT, CHANGE, or BROWSE commands. The form is stored as a format (.fmt) file. Format files created using dBASE III PLUS have an .fmt extension; these files are compiled with SET FORMAT TO in dBASE IV to .fmo files.

## Syntax

```
SET FORMAT TO [ <format filename> /?]
```

## Usage

You can use CREATE/MODIFY SCREEN to create format files. CREATE/ MODIFY SCREEN first creates a screen .scr file. Make sure you select **Save this form** from the **Layout** menu to create the .fmt file. This format file can be used with SET FORMAT TO when you are editing or changing records in the associated database file.

The SET FORMAT TO command activates the format file named in the command for use with full screen editing commands.

The SET FORMAT TO command looks for a compiled format file with the .fmo extension. If it does not find one, then it looks for a format file with an .fmt extension. The first time you use an .fmt file, a compiled version of it is created with a .fmo extension. From that point on, the .fmo file is used with SET FORMAT TO. If neither an .fmt, nor an .fmo file can be found, you cannot use the specified format file.

When you want to change a format file, use the MODIFY SCREEN command because this command updates both the .scr and .fmt files. Do not use the MODIFY COMMAND text editor to modify the format file, because the changes will not be made to the screen file and the two files will no longer correspond to the same format.

A format file is divided into three sections. The first section is the initialization and can contain any command except @ commands, READ, or any commands that deactivate the format file. You can create memory variables in this section which you can use in the following two sections of the format file.

The second section is the forms control part of the format file and can contain only @ commands and a READ command to denote the end of a form page.

The third section is the termination and can contain the same type of commands as the initialization section. Any memory variable created in the initialization of the format file can be released in the termination section.

**NOTE**
*Do not include a READ command at the end of the .fmt file. dBASE IV interprets this as a page break, and the cursor will stop in the corner of the screen waiting for a keypress.*

Format files that contain the new keywords to the @ command (such as VALID or WHEN) are not backward compatible with dBASE III PLUS.

If SET FORMAT TO is not used, APPEND, CHANGE, EDIT, and INSERT use the standard display and entry form dBASE IV provides.

The SET FORMAT TO command updates a catalog if one is open, and if SET CATALOG is ON.

If a catalog is open, SET FORMAT TO ? displays a directory of all files with the .fmt extension for the active database file. If a catalog is not open, then SET FORMAT TO ? displays a list of files on the current drive.

Each work area with an open database file can have an open format file. Both the SET FORMAT TO and the CLOSE FORMAT commands close the open format file in the currently selected work area.

## See Also

@, APPEND, CHANGE, CLOSE, EDIT, INSERT, READ, SET DEVICE

# SET FULLPATH

dBASE IV functions that return filenames return the full file and pathname, whereas the same functions return only the drive and the filename in dBASE III PLUS. The SET FULLPATH command makes dBASE III PLUS programs that contain the functions MDX(), NDX(), and DBF() dBASE IV compatible by suppressing the display of the full pathname.

## Syntax

SET FULLPATH on/OFF

## Default

The default for SET FULLPATH is OFF.

## Usage

Use this command with dBASE III PLUS programs that use the functions MDX(), NDX(), and DBF(), to suppress the return of the full file specification.

You can use this command from the dot prompt, or you can add it to your Config.db file to always suppress the pathname display. FULLPATH is not part of the full-screen **SET** menu, and cannot be changed from that menu.

## See Also

See *dBASE IV Change Summary* for a discussion of changes and enhancements introduced in dBASE IV.

# SET FUNCTION

SET FUNCTION programs a function key. You may associate up to 238 characters with each function key. Thereafter, each time you press that function key, the characters programmed to that key are transmitted to the current input operation.

## Syntax

```
SET FUNCTION <expN>/<expC>/<key label> TO <expC>
```

## Usage

You may program the function keys by using three different approaches:

1. Use the **Keys** submenu of the **SET** menu and assign new functions.
2. Use the SET FUNCTION command at the dot prompt or from within a program.
3. Use the Config.db file to assign new functions to programmable keys. See Chapter 6, "Customizing dBASE IV."

The standard function key assignments are listed in Table 3-4.

Table 3-4 Default function key assignments

| Key | Value | Key | Value |
|---|---|---|---|
| F1 | HELP; | F6 | DISPLAY STATUS; |
| F2 | ASSIST; | F7 | DISPLAY MEMORY; |
| F3 | LIST; | F8 | DISPLAY; |
| F4 | DIR; | F9 | APPEND; |
| F5 | DISPLAY STRUCTURE; | F10 | EDIT; |

Function key **F1** is assigned to the Help function and cannot be reprogrammed.

The programmable function keys are **F2** through **F10**, **Shift-F1** through **Shift-F9**, and **Ctrl-F1** through **Ctrl-F10**.

**Shift-F10** and all **Alt** key combinations are macro keys and cannot be programmed.

If you have an enhanced keyboard with function keys **F11** and **F12**, these keys are not programmable in dBASE IV.

## Examples

Put a semicolon at the end of the text string of functions and commands that you assign to a function key so that the function executes when you press the programmed key or key combination. Use quotes to delimit the beginning and the end of the text string.

To assign the SET command to function key **F10**, so that pressing **F10** executes the command:

```
. SET FUNCTION F10 TO "SET;"
```

To assign multiple commands to function key **F9**, so that the commands CLEAR, USE Client, and LIST STRUCTURE are executed:

```
. SET FUNCTION F9 TO "CLEAR;USE Client;LIST STRUCTURE;"
```

**NOTE**
*SET FUNCTION is overridden by ON KEY in programs. If SET FUNCTION and ON KEY assign different commands to the same key, the ON KEY assignment overrides that of SET FUNCTION.*

# SET HEADING

SET HEADING determines whether or not column titles are shown above each field for DISPLAY, LIST, SUM, and AVERAGE.

## Syntax

SET HEADING ON/off

## Default

The default for SET HEADING is ON.

## Usage

The commands DISPLAY, LIST, SUM, TYPE, and AVERAGE display a column title for each displayed field, memory variable, or expression.

The column width is the length of the heading or field width, whichever is larger.

## Examples

```
. USE Transact
. DISPLAY

Record#  Client_id   Order_id   Date_trans   Invoiced   Total_bill
      1  A10025      87-105     02/03/87     .T.           1850.00
```

```
. SET HEADING OFF
. DISPLAY

      1  A10025      87-105     02/03/87     .T.           1850.00
```

# SET HELP

SET HELP determines whether a pop-up window with dBASE IV Help options appears, when a command is incorrectly entered at the dot prompt.

## Syntax

SET HELP ON/off

## Default

The default for SET HELP is ON.

## Usage

If you make an error entering a command, the available Help options appear inside a pop-up window. You can select to CANCEL the command, to EDIT the command syntax on the command line, or to go to the Help system and look up the command syntax.

## See Also

HELP, SET INSTRUCT

# SET HISTORY

SET HISTORY stores executed commands in a history buffer. SET HISTORY TO specifies the number of executed commands that are stored in the history buffer.

## Syntax

SET HISTORY ON/off

SET HISTORY TO <expN>

## Defaults

The default for SET HISTORY is ON, and for SET HISTORY TO is 20. You can change SET HISTORY TO to any value between 0 and 16,000.

## Usage

SET HISTORY allows you to redisplay, edit, or re-execute commands executed from the dot prompt. Commands executed inside a program file are not stored in the history buffer.

Use ↑ and ↓ to redisplay commands you previously executed from the dot prompt. Commands are displayed from most recent to least recent as you step through the history buffer using the ↑ key. Commands stored in the history buffer reside in memory. You can LIST HISTORY or DISPLAY HISTORY to view all the commands in the buffer.

The number of commands the history buffer stores is determined by SET HISTORY TO. When this number is reached, the earliest commands executed are cleared from the buffer. Decreasing the value of SET HISTORY TO causes the number of stored commands to be decreased to the new value. SET HISTORY TO 0 clears all commands from the history buffer.

The maximum size of the history buffer is limited by the amount of memory you have available, and by the memory occupied by other programs already resident in memory, including dBASE IV. To calculate the amount of memory that the history buffer uses, add nine bytes to the number of bytes in each command.

## See Also

DISPLAY HISTORY, LIST HISTORY

# SET HOURS

The SET HOURS command changes the time display to either a 12-hour or 24-hour clock.

## Syntax

SET HOURS TO [12/24]

## Default

The default for SET HOURS is 12.

## Usage

When you use the SET HOURS TO command to change the display format, the clock display in all full screen commands is changed.

SET HOURS TO changes the clock screen display for all screens that display the system clock such as BROWSE or Control Center.

The only allowable arguments for this command are 12 and 24. You may enter this command in the Config.db file to make the clock always display 24 hours. See Chapter 6, "Customizing dBASE IV."

## Examples

At 8:00 p.m., the clock display changes as follows:

```
. SET HOURS TO 12
8:00:00 pm

. SET HOURS TO 24
20:00:00
```

To return to the default:

```
. SET HOURS TO ↵
```

## See Also

SET CLOCK

# SET INDEX

SET INDEX opens the specified index files, both index (.ndx) and multiple index (.mdx). It may optionally specify the controlling index or tag for an active database file.

## Syntax

SET INDEX TO [?/<filename list> [ORDER [TAG] <.ndx filename> /
    <.mdx tagname> [OF <.mdx filename>]]]

## Defaults

If you do not specify a file extension, SET INDEX attempts to open an .mdx file first, then an .ndx file.

SET INDEX TO first closes all open indexes in the current work area except the production .mdx file. Then it opens the indexes specified in the filename list.

SET INDEX TO ↵ closes all open .ndx files in the current work area. It is an alternate syntax for CLOSE INDEX. The .mdx files are left open.

## Usage

The index given in the ORDER clause is the master or controlling index. This index may be either an index (.ndx) file or a tag name contained in a multiple index (.mdx) file. The master index controls the movement of the record pointer in the database file.

The FIND and SEEK commands use the master index to locate matching records. All other open indexes will be updated when data in their keys change, but they do not control the record pointer.

You can optionally state the .mdx file which contains the tag name by using the OF <.mdx filename> phrase. If a tag name is contained in an .mdx file other than the production .mdx file, or if two open .mdx files contain the same tag name, you should use the OF phrase to indicate the correct index.

If only one index is open, that index controls the index order; the ORDER clause is not needed.

Using the SET ORDER command, you can switch the index file designated as the master index without changing the open indexes or .mdx files.

All open indexes, including the master index, are automatically updated whenever a change is made to the associated database file. All tag names within open .mdx files are also updated.

## Record Pointer

When you issue SET INDEX TO, the database record pointer is positioned at the beginning of the file as determined by the master index file.

## Special Cases

Although commands that access the database file show the records arranged in the master index order, the physical order of the records in the file on disk is not changed. If you restore the natural order of the database file with SET ORDER TO 0 or SET ORDER TO ↵, all indexes will continue to be updated. If you restore natural order with SET INDEX TO ↵, the production .mdx file will continue to be updated, but no other indexes will be updated. You will subsequently need to open and REINDEX all indexes, except for the production .mdx index.

If you APPEND records to a database file that uses an index, new records are added to the end of the physical database file. After you complete the field entries in a new record, the record's key expression is included in the master index, and the record assumes its position in indexed order. All open indexes are updated, as well as all tags within an open .mdx file.

The INSERT command is equivalent to APPEND when an index file is in use, and will add records to the end of the physical database file.

dBASE IV allows a maximum of 47 .mdx tag names or 10 .ndx files plus the production .mdx file to be open simultaneously. When an .mdx file is opened, all tags in that file are also opened.

## Examples

Open the Cus_name index file when the Client database file is open. Type:

```
. USE Client
. SET INDEX TO Cus_name
Master index: Cus_name
```

Open the Cus_name index file and set the master index to the Client tag in the production .mdx file. Type:

```
. SET INDEX TO Cus_name ORDER Client
Master index: Client
```

## See Also

CLOSE, DELETE TAG, DISPLAY INDEXES, INDEX, KEY(), MDX(), NDX(), ORDER(), REINDEX, SET ORDER, TAG(), USE

# SET INSTRUCT

SET INSTRUCT enables the display of prompts.

## Syntax

SET INSTRUCT ON/off

## Default

The default for SET INSTRUCT is ON. When ON, this command enables the prompts of the dBASE IV menu interface. It works with all full-screen operations, such as APPEND, BROWSE, and EDIT. When INSTRUCT is OFF, no prompts are displayed.

# SET INTENSITY

SET INTENSITY determines whether the enhanced screen attribute is used for full-screen operations such as APPEND and EDIT.

## Syntax

SET INTENSITY ON/off

## Default

The default for SET INTENSITY is ON.

## Usage

When SET INTENSITY is ON, dBASE IV displays @...SAY commands with the *standard* screen attribute and displays @...GET commands using the *enhanced* screen attribute.

You can assign different display attributes to standard and enhanced displays with the SET COLOR command, or by using the full-screen SET command. The full-screen SET command provides an easy way to see your selections as you make them.

If SET INTENSITY is OFF, the enhanced screen attribute is not used. Instead, the standard attribute is used for both @...GET and @...SAY commands.

## See Also

SET, SET COLOR, SET DISPLAY

# SET LOCK

SET LOCK determines whether a lock is applied to records in multi-user systems to prevent a record from being updated by more than one user at the same time.

## Syntax

SET LOCK ON/off

## Default

The default for SET LOCK is ON.

## Usage

Locks are applied to records or files to prevent data collisions which can result when more than one user attempts to write to a file or record at the same time. Locks permit read-only access by any user to files and records, even when they may be in the process of being updated. Files and records in use are automatically locked when you use any command that updates or edits files and records.

If you SET LOCK OFF, you can disable automatic locking for a read-only subset of the commands listed in Table 3-5. These commands work without locking, but data integrity is not guaranteed. Some of these commands have two phases: reading and writing. For these commands, SET LOCK OFF applies only to the reading phase. When writing begins, the file is LOCKed.

Table 3-5 Commands that automatically LOCK files

| dBASE Command | Action | Level | Does SET LOCK OFF Disable LOCK? |
|---|---|---|---|
| @GET/READ | edit | Record | No |
| APPEND FROM | update | File | No |
| APPEND [blank] | update | Record | No |
| AVERAGE | read only | File | Yes |
| BROWSE | edit | Record | No |
| CALCULATE | read only | File | Yes |
| CHANGE/EDIT | edit | Record | No |
| COPY TAG/INDEX | read/write | File | Yes on read; No on write |
| COPY [STRUCTURE] | read/write | File | Yes on read; No on write |
| COUNT | read only | File | Yes |
| DELETE/RECALL | update | Record | No |
| DELETE/RECALL | update | File | No |
| EDIT | update | Record | No |
| INDEX | read/write | File | Yes on read; No on write |
| JOIN | read/write | File | Yes on read; No on write |
| LABEL | read only | File | Yes |
| REPLACE | update | Record | No |
| REPLACE [scope] | update | File | No |
| REPORT | read only | File | Yes |
| SET CATALOG ON | catalog | File | No |
| SORT | read/write | File | Yes on read; No on write |
| SUM | read only | File | Yes |
| TOTAL | read/write | File | Yes on read; No on write |
| UPDATE | update | File | No |

When command execution is complete, the file or record is unlocked.

A record must be locked before it can be updated. In dBASE IV, records are locked automatically when you press any key other than a cursor positioning key. dBASE IV supports the dBASE III PLUS locking key combination, that is, **Ctrl-O**.

dBASE IV also continues to support control key combinations that are used to unlock records in dBASE III PLUS, such as **Ctrl-C** or **Ctrl-R**.

In dBASE IV, any time you move the cursor to another record (previous or next) the record you moved from is automatically unlocked. The exception to this is the REPLACE command. (See "Special Cases" below.)

## Special Cases

If the scope of the REPLACE command is a RECORD, then the lock is not released until you execute a command other than REPLACE. If the next command is another replace, the record remains locked. This ensures that all replace statements for a given record are complete before the record is released to other users.

BROWSE and EDIT use an automatic record lock on only the record that is being modified, allowing other users access to all the other unlocked records. This feature is useful when several users have to check or update records from the same database file; for example, when many agents need access to customer records in one database file.

In dBASE IV, SET LOCK does not automatically release locks when multiple records are locked using the RLOCK() function. To unlock these, issue an UNLOCK command.

# SET MARGIN

SET MARGIN adjusts the printer offset for the left margin for all printed output. The video display is unaffected.

## Syntax

SET MARGIN TO <expN>

## Default

The default for the left margin is zero.

## Usage

SET MARGIN TO sets the margin for all printed output from dBASE IV.

The SET MARGIN value is the same as the system memory variable _ploffset. Both of these commands move the left margin. They are not additive; the last executed command is the one in effect.

You can include the SET MARGIN command in the Config.db file, to assign an inital value to the printer left offset. See Chapter 6, "Customizing dBASE IV."

The left margin setting specified with the _lmargin system memory variable is added to the SET MARGIN setting when _wrap is true. When _wrap is false, _lmargin and SETMARGIN are not additive.

## Examples

To set the left margin 10 spaces to the right of the first possible print position:

```
. SET MARGIN TO 10
. ? _ploffset
10
```

The margin setting can be changed with _ploffset:

```
. _ploffset=5
5
```

The last change that was made with _ploffset is now also the margin setting. If you do a LIST STATUS command, you will see that the margin setting is now equal to 5. Until this setting is changed, all output sent to the printer will be spaced 5 spaces to the right of the first possible print position.

# SET MARK

SET MARK changes the delimiter character used to separate the month, day, and year in the date display.

## Syntax

SET MARK TO [ < expC > ]

## Usage

< expC > is a single character *delimited by quotes* to be used in separating the month, day, and year when displaying dates. The default changes by country; for the U.S., it is a slash (/). To restore the default delimiter character, type:

```
. SET MARK TO ↵
```

## Example

```
. SET MARK TO "."
. ? DATE()
01.22.88
```

## See Also

DATE(), DMY(), MDY(), SET CENTURY, SET DATE

# SET MEMOWIDTH

SET MEMOWIDTH adjusts the width of memo field output.

## Syntax

```
SET MEMOWIDTH TO <expN>
```

## Default

The default memo width is 50 characters; the minimum is 8, and the maximum is 32,000. You may customize this value from the Config.db file. See Chapter 6, "Customizing dBASE IV."

## Usage

Use SET MEMOWIDTH TO to alter the column width of memo fields during output. If the system memory variable _wrap is set to true (.T.), the system variables _lmargin and _rmargin determine the memo width.

The @V (vertical stretch) picture function causes memo fields to be displayed in a vertical column when _wrap is true. When @V is equal to zero, memo fields word wrap within the SET MEMOWIDTH width. When @V is specified, the _pcolno system memory variable is incremented by the @V value. This allows you to change the appearance of the printed output of ?/?? commands by using the @V function.

## See Also

MEMLINES(), MLINE(), @V PICTURE FUNCTION, SET()

# SET MENU

The SET MENU command applies to dBASE III PLUS only. It determines whether a menu of cursor movement keys appears with the full-screen commands.

## Syntax

SET MENU ON/off

## Default

In dBASE III PLUS, the default for SET MENU is ON. SET MENU has no effect in dBASE IV.

# SET MESSAGE

SET MESSAGE displays a user-defined character string on the bottom line of the screen if SET STATUS is ON.

## Syntax

SET MESSAGE TO [ < expC > ]

## Default

The default is to display on dBASE IV messages. SET MESSAGE TO ↵ resets the message to the default. The user-defined message does not display unless SET STATUS is ON.

## Usage

The character string can have a maximum of 79 characters. If you specify a character string, it must be delimited with any valid dBASE IV delimiter (' ', " ", or [ ]). You can also specify a character type memory variable or display a field which contains the literal text string.

In full-screen menu programs, such as ASSIST, you can't set your own message; the application messages override it.

The message line is in effect, as long as SET STATUS is ON, when you are entering commands from the dot prompt and with full-screen editing commands such as APPEND, EDIT, and READ, or from within a program. The message is displayed centered on line 23 of the screen, below the status bar.

Messages from @...GET commands, POPUPs, and MENUs overwrite the message from SET MESSAGE.

## Tips

If you are using the status bar and message line, you should not specify any line below 21 in your format file. When SET STATUS is ON, you can use SET MESSAGE to convey useful information on line 23.

## See Also

SET STATUS

# SET NEAR

SET NEAR positions the database file to the record immediately following the potential location of the sought-after key in that file.

## Syntax

SET NEAR on/OFF

## Default

The default for SET NEAR is OFF.

## Usage

When SET NEAR is ON, the database record pointer is set to the record nearest the key expression that was searched and not found. When SET NEAR is OFF, the database is positioned at the end of the file when a search is unsuccessful.

When SET DELETED or SET FILTER is ON, SET NEAR uses either or both of these commands in selecting the record nearest to the key expression. It does not use records that are marked for deletion, and it follows the SET FILTER TO restrictions.

## Example

To find a Client_id beginning with "D" from the Customer database file, or to find the first record after the position where "D" would occur:

```
. USE Client ORDER Client_id
Master index: CLIENT_ID
. LIST Client_id

Record#     Client_id
      1     A00001
      5     A00005
      8     A10025
      7     B12000
      3     C00001
      2     L00001
      4     L00002

. SET NEAR ON
. FIND D
Find not successful.
. ? EOF()
.F.
. ? FOUND()
.F.
```

You can no longer use the EOF() function to see if an exact match has occurred; use FOUND() to determine exact matches with SET NEAR ON.

With SET NEAR ON:
FOUND() returns a true (.T.) if an exact match has occurred.
FOUND() returns a false (.F.) for a near match, and the database file is positioned at the record whose key sorts immediately next to the sought value.
EOF() returns a false (.F.) if the match is a near match.

With SET NEAR OFF:
FOUND() returns a false (.F.) if no match occurs.
EOF() returns a true (.T.) if no match occurs; however, this function is returning a true (.T.) because the database file is positioned to the end-of-file.

## See Also

EOF(), FIND, FOUND(), LOCATE, SEEK, SEEK()

# SET ODOMETER

SET ODOMETER defines the update interval of the record counter for commands that display a record count.

## Syntax

SET ODOMETER TO <expN>

## Default

The default interval is one, and the maximum is 200.

## Usage

SET ODOMETER determines the interval at which the record counter is updated on the screen for commands such as COPY and RECALL.

Using SET TALK OFF, you can remove the record counter from the screen entirely.

# SET ORDER

SET ORDER sets up any open index file or tag as the master (controlling) index, or removes control from all open indexes, without closing any .mdx or .ndx files.

## Syntax

```
SET ORDER TO
SET ORDER TO <expN>
SET ORDER TO [TAG]
    <filename> / <.mdx tagname> [OF <.mdx filename>]
```

## Usage

This command saves time when you want to reassign a controlling index file, because it does not close and open the index files or move the record pointer. SET ORDER TO command has three modes.

SET ORDER TO without an argument restores the database file to its natural order, and can be used with both .mdx or .ndx files.

SET ORDER TO <expN> provides dBASE III PLUS compatibility. This mode may be used only with .ndx files, when there are no open .mdx files. The numeric expression is limited to a number between 0 and 10 corresponding to the number of open .ndx files you have. The number you specify corresponds to the position of a file in the list of index files specified by SET INDEX TO or with the USE command.

If you have both .mdx and .ndx files open, use the third mode to assign controlling index files or tags. No numeric expression may be used when both types of index files are open; you must specify the ORDER by its filename. The ORDER filename may be an index filename, or an indirect file reference that can be interpreted as a filename. The number of the open .ndx file in the list cannot be used.

All open index files are updated if the index key is changed or if records are added or packed. SET ORDER TO 0 leaves the .ndx file open for updating.

If the tag is contained in an .mdx file other than the production .mdx file, or if an identical tag name is in more than one open .mdx file, you should include the OF <.mdx filename> phrase and indicate the .mdx file you want to use.

Commands which do not take advantage of index files, such as LOCATE, operate faster if no indexes are open. Issue a SET INDEX TO before issuing any command that repositions the record pointer without using the index.

### Examples

When the database file, Customer, is opened, the master index may be specified by the TAG Client with the ORDER option of USE. To change the master index from the TAG Client to the Cus_name index file:

```
. USE Client INDEX Cus_name ORDER Client
Master index: CLIENT
. SET ORDER TO TAG Cus_name
Master index: CUS_NAME
. SET ORDER TO
Database is in natural order
```

To set the master index back to the TAG Client:

```
. SET ORDER TO Client
Master index: CLIENT
```

### See Also

DISPLAY, INDEX, KEY(), MDX(), NDX(), ORDER(), REINDEX, SET INDEX, TAG(), USE

# SET PATH

SET PATH specifies the directory search route that dBASE IV follows to find files that are not in the current directory. dBASE IV does not use the path established by the DOS PATH command, nor does DOS use the path that dBASE IV establishes with SET PATH. Also, the Control Center does not use the path set with the SET PATH command.

## Syntax

SET PATH TO [ <path list> ]

A *path list* is a series of directory names separated by commas or semicolons and up to 60 characters long, which specifies the directory search path.

## Defaults

The default is the current directory. SET PATH TO ↵ without a path list restores the default path.

## Usage

The default directory is where you started dBASE IV from, unless you changed directories using the DOS CHDIR command or used the SET DEFAULT command to specify a different drive.

SET PATH does not affect the DIR command listing. The DIR command lists files in the current or specified directory, not in the directory set through a path. To check a directory other than the current one, use the DIR command and give the path. For example, to search DBDATA type:

```
. DIR \DBASE\DBDATA\
```

If you want to create a new file in a directory other than the current directory, specify the full path name along with the filename to the CREATE command.

## Example

dBASE IV first searches for the file Program.dbo in the C:\DBASE\DBDATA subdirectory. If it is not found, dBASE IV will search the default directory where dBASE IV was started.

```
. SET PATH TO C:\dBASE\DBDATA
. DO Program
```

## See Also

DIR, SET DEFAULT

# SET PAUSE

SET PAUSE controls the display of the SQL SELECT command, and stops the display of data after each screenful.

## Syntax

SET PAUSE on/OFF

## Default

The default for SET PAUSE is OFF.

## Usage

When SET PAUSE is ON, the SQL SELECT command operates similarly to the dBASE IV DISPLAY command; it sends a line of data to the screen in response to each DISPLAY command until the screen is full. This provides a method for producing multi-screen displays that pause between screen loads, and column headings that can be called multiple times from dBASE IV code.

When SET PAUSE is OFF, the SQL SELECT command operates similarly to the dBASE IV LIST command; it provides column headings before the first record only, then it prints the rest of the records without pausing between screen loads.

## See Also

See *Using dBASE IV SQL* for more information on SQL commands.

# SET POINT

SET POINT changes the character used for the decimal point.

## Syntax

SET POINT TO [ < expC > ]

## Default

The default for SET POINT is a period (.).

## Usage

Use this command to change the decimal point from the period (.) to the comma (,) for international usage. You can also specify any quoted single alphanumeric character. You may not use a digit or a space for the decimal point.

To restore the default, which is a period (.), type SET POINT TO ↵.

# SET PRECISION

SET PRECISION determines the number of digits that dBASE IV uses internally in all math operations that use type N numbers.

## Syntax

SET PRECISION TO [ < expN > ]

where < expN > is an integer between 10 and 20.

## Default

The default for SET PRECISION is 16. The range of the optional numeric expression can be from 10 to 20.

## Usage

SET PRECISION TO specifies the number of accurate digits used in mathematical operations.

The default value for SET PRECISION TO is 16. This may be changed from the dot prompt or from the Config.db file. See Chapter 6, "Customizing dBASE IV."

## See Also

SET DECIMALS

# SET PRINTER

SET PRINTER ON directs all output not formatted with the @...SAY command to the printer or to a file.

SET PRINTER TO redirects output to a local printing device, to a shared network printer, or to a file. On a network, SET PRINTER TO also signals the file server to print the next printjob.

## Syntax

This command has six forms:

SET PRINTER on/OFF

The default is OFF.

SET PRINTER TO <DOS device>

The default is DOS print utility PRN.

SET PRINTER TO \\<computer name>\<printer name> =
<destination>

SET PRINTER TO \\SPOOLER

SET PRINTER TO \\CAPTURE

SET PRINTER TO FILE <filename>

## Usage

The first five forms of this command are used to direct or redirect printer output to different print devices. The sixth syntax produces printer formatted files.

### Redirecting Screen Output to Printer

When SET PRINTER is ON, all screen output, including TALK, LIST, and ? is routed to the printer.

Use SET DEVICE TO PRINTER to direct @...SAY commands to the printer. SET PRINTER ON does not affect @...SAY commands.

### Printing to a Local Device

Use SET PRINTER TO <DOS device> to send printed output to a local printer. The default is the DOS print utility PRN. You can assign this to any one of the three parallel ports (LPT1, LPT2, LPT3) or one of the two serial ports (COM1 or COM2). For example:

```
. SET PRINTER TO LPT2
```

sends print output to the printer connected to parallel port 2. Once you change the default from PRN to a DOS device, you cannot go back to PRN again.

The SET PRINTER TO command is used to reset the DOS default device. In a network environment, dBASE IV signals the file server to execute the most recent printjob. Therefore, you can use the SET PRINTER TO < DOS device > command to initiate spooling and reset the default DOS device. This form of the command empties the print spooler and resets the print destination to its default, or to another destination. For example:

```
. SET PRINTER TO LPT1
```

resets the print output to the local parallel printer.

## Printing on a Network

To use the IBM PC and 3Com networks:

```
. SET PRINTER TO \\<computer name>\<printer name> = <destination>
```

< computer name > is the network-assigned name for the network file server.

< printer name > is the network-assigned name for the printer used on the IBM network. There may be more than one of each on a given network.

< destination > identifies the installed shared printer as LPT1, LPT2, or LPT3. The shared printer may be at the logged user's workstation or at a remote workstation. For example:

```
. SET PRINTER TO \\SERVER\EPSON=LPT1
```

sends output to a shared Epson printer off the file server designated as LPT1 on that server.

Each time you direct print output to a different printer or spooler, you can reset the default by setting the printer to the default printer name:

```
. SET PRINTER TO <DOS default>
```

## Printing to a File

This command is used to produce a printer-formatted file that you can send to the printer for which it was formatted. If this file is produced with the print driver Ascii.pr2, then you get an ASCII text file that is suitable for sending to any line printer, or through telecommunications links.

To create a printer formatted file, type:

```
. SET PRINTER TO FILE <filename>
```

dBASE IV assigns the default extension of .prt to files formatted for the printer, unless you use the Ascii.pr2 print driver, in which case they are assigned the default extension of .txt.

To get a text file, assign the Ascii.pr2 print driver by typing:

```
. _pdriver = "Ascii.pr2"
```

Now, when you send the output to a file, the result will be an ASCII file.

To get a file formatted for a specific printer, for example, the Hewlett-Packard LaserJet, assign the print driver by typing:

```
. _pdriver = "Hplas100.pr2"
```

Next, select the file you would like to print on the LaserJet and send the output to a file. Type:

```
. SET PRINTER TO FILE Filename
```

The resulting file will be Filename.prt and will be imbedded with the printer control codes for the LaserJet. This file can be sent directly from the DOS prompt of any PC connected to a LaserJet printer.

Assuming the printer formatted file, Filename.prt, is on a diskette in the A drive and the LaserJet is attached to serial port COM1, from the default DOS prompt type:

```
COPY A:Filename.prt COM1
```

The printed output will be formatted correctly for the LaserJet. Notice that you are copying a file and not using the DOS print utility. Do not use the DOS PRINT command, as this will print the imbedded control codes along with the text and produce unusable output.

SET PRINTER TO FILE remains in effect until you issue a SET PRINTER TO command and redirect output to the local DOS device or to the network spooler. When you restore the default device, you might need to change the print driver used with SET PRINTER TO FILE if this is different from the default driver. Change your printer driver by setting _pdriver equal to the name of the print driver you want.

## Examples

The following commands spool the output of report Names to the network printer (parallel printer number 1) attached to the server named ASERVER:

```
. SET PRINTER TO \\ASERVER\PRINTER=LPT1
. REPORT FORM Employee TO PRINTER
. SET PRINTER TO LPT1
```

The first command redirects LPT1 to the network printer; the second sends the output to the spooler; the third sends the spooled output to the printer and resets the network printer.

## See Also

?/??, ???, LABEL FORM, LIST/DISPLAY, REPORT FORM, SET DEBUG, SET DEVICE

# SET PROCEDURE

SET PROCEDURE opens a named procedure file.

## Syntax

SET PROCEDURE TO [ < procedure filename > ]

## Defaults

The procedure filename must include the drive location if the named file is not on the default drive or in a dBASE search path.

If you do not specify a file extension for the procedure filename, dBASE IV assumes an object procedure file with the .dbo extension. Compiled dBASE IV procedure files use the .prc extension and dBASE IV/SQL procedure files use the .prg extension. If dBASE IV cannot find the specified file, it will compile the source file to produce a .dbo file.

SET PROCEDURE TO < procedure filename > opens the specified procedure file. SET PROCEDURE TO ↵ or CLOSE PROCEDURE closes the current procedure file.

## Usage

A procedure file may contain a theoretical maximum of 1,170 procedures (routines); however, available RAM may limit this number. The command PROCEDURE marks the beginning of each new routine. Only one procedure file may be open at a time.

Any application can have procedure files in it, but only one SET PROCEDURE can be in effect at a time.

## Examples

The following program and procedure files illustrate the use of procedures. For example, you could create the following program which opens a procedure file and executes the two procedures within that file:

```
* Setproc.PRG - Demonstrates SET PROCEDURE TO

SET PROCEDURE TO Proctest        && Open the procedure file.
DO Proc1                         && Execute procedure 1.
DO Proc2                         && Execute procedure 2.
CLOSE PROCEDURE                  && Close the procedure file.
* EOP: Setproc.prg
```

The setup of the Proctest procedure file (which is also a dBASE program file) could be as simple as the following:

```
* Proctest.prc - Procedures
PROCEDURE Proc1
   ? "This is a message from Proc1 of Proctest.prc."
RETURN
* EOP: Proc1

PROCEDURE Proc2
   ? "This message is in Proc2 of Proctest.prc."
   DO Proc3
RETURN
* EOP: Proc2

PROCEDURE Proc3
   ? "This message is in Proc3 of Proctest.prc and"
   ? " was called from Proc2 of Proctest.prc."
RETURN
* EOP: Proc3
* EOP: Proctest.prc
```

To execute the procedure (if both the dBASE program file and the procedure file are in the default directory or along the path specified by SET PATH TO), you would type:

```
. DO Setproc
This is a message from Proc1 of Proctest.prc.
This message is in Proc2 of Proctest.prc.
This message is in Proc3 of Proctest.prc and
 was called from Proc2 of Proctest.prc.
```

## See Also

COMPILE, DO, PARAMETERS

# SET REFRESH

SET REFRESH specifies the time interval between file checks, to determine if a record has changed in a file that is being BROWSEd or EDITed on a network. This command works only on files that have been converted with the CONVERT command.

## Syntax

SET REFRESH TO <expN>

## Default

The default for SET REFRESH is zero.

## Usage

You can enter this command from the dot prompt before beginning a BROWSE or EDIT session, or set the refresh interval from Config.db. At the end of each refresh interval, dBASE IV checks all records that are being BROWSEd on the network and displays any updated information.

If the file is being BROWSEd and the refresh count reaches zero, dBASE IV checks the files that are being BROWSEd on the network and updates the screen with any changes it finds.

The numeric expression is a time interval expressed as seconds. The range is from 1 to 3,600 (one hour), and the default is 0.

If the refresh interval is 0, the system does not refresh.

## See Also

CONVERT

# SET RELATION

SET RELATION links two open database files according to a key expression that is common to both files.

## Syntax

SET RELATION TO

SET RELATION TO <exp>

SET RELATION TO <expN> INTO <alias>
    [, <expN> INTO <alias>]...

## Defaults

SET RELATION TO ↵, without any additional parameters, removes the relation from the currently selected work area.

## Usage

SET RELATION links the active database file to an open database file in another work area. The INTO file is identified by its alias. The active database file that controls the relation is referred to as the *parent* file. The file which is linked to the parent file by a relating expression is referred to as the *child*.

## Options

With the numeric expression option, the active database is always linked to the record number in the file specified by the numeric expression. Typically, the numeric expression will be the RECNO() function. This causes record 1 in the parent file to be linked to record 1 in the child, record 2 in the parent to record 2 in the child, and so on.

## Record Pointer

If a matching record cannot be found in the linked file, the linked file is positioned at the end of the file (a phantom record), and EOF is true (.T.).

The child files of SET RELATION do not honor SET NEAR.

## Tips

You can save a working environment, including relations, with CREATE VIEW <.vue filename> FROM ENVIRONMENT. Remember that CREATE/MODIFY VIEW is available to you as a menu-driven command for linking files. It also allows you to save relations to a .vue file.

## Examples

To relate the Transact database file into the Customer database file, link the two files on the Client_id:

```
. SELECT 1
. USE Client ORDER Client_id
Master index: CLIENT_ID
. SELECT 2
. USE Transact
. SET RELATION TO Client_id INTO Client
. GO TOP
. LIST NEXT 5 Client_id, Client->Client, Date_trans, Total_bill

Record#   Client_id   Client->Client          Date_trans  Total_bill
      1   A10025      PUBLIC EVENTS           02/03/87       1850.00
      2   C00001      L. G. BLUM & ASSOCIATES 02/10/87       1200.00
      3   C00002      TIMMONS & CASEY, LTD    02/12/87       1250.00
      4   C00001      L. G. BLUM & ASSOCIATES 02/23/87       1250.00
      5   L00001      BAILEY & BAILEY         03/09/87        415.00
```

To relate a database file into multiple database files, continue with the environment from the previous example:

```
. SELECT 3
. USE Stock ORDER Order_id
Master index: ORDER_ID
. SELECT 2
. SET RELATION TO Client_id INTO Client, Order_id INTO Stock
. GO TOP
. LIST NEXT 5 Client->Client, Date_trans, Stock->Part_name

Record#     Client->Client          Date_trans   Stock->Part_name
      1     PUBLIC EVENTS           02/03/87     SOFA, 6-FOOT
      2     L. G. BLUM & ASSOCIATES 02/10/87     SOFA, 8-FOOT
      3     TIMMONS & CASEY, LTD    02/12/87     CHAIR, DESK
      4     L. G. BLUM & ASSOCIATES 02/23/87     CHAIR, DESK
      5     BAILEY & BAILEY         03/09/87     TABLE, END
```

To establish a chain relation of the Items database file into the Orders database file into the Customer database file, continue with the previous example's environment:

```
. SET RELATION TO Client_id INTO Client
. SET ORDER TO Order_id
Master index: ORDER_ID
. SELECT 3
. SET INDEX TO           && Set the file back to natural order.
. SET RELATION TO Order_id INTO Transact
. GO TOP
. LIST NEXT 5 Part_name, Transact->Date_trans, Client->Client

Record#  Part_name      Transact->Date_trans   Client->Client
      1  SOFA, 6-FOOT   02/03/87               PUBLIC EVENTS
      2  SOFA, 6-FOOT   02/03/87               PUBLIC EVENTS
      3  SOFA, 8-FOOT   02/10/87               L. G. BLUM & ASSOCIATES
      4  CHAIR, DESK    02/12/87               TIMMONS & CASEY, LTD
      5  CHAIR, DESK    02/23/87               L. G. BLUM & ASSOCIATES
```

## See Also

CREATE/MODIFY VIEW, SET FIELDS, SET SKIP, SET VIEW

# SET REPROCESS

SET REPROCESS sets the number of times dBASE IV tries a network file or record lock command before producing an error message.

## Syntax

SET REPROCESS TO <expN>

## Default

The default is zero. The range is from −1 to 32,000.

## Usage

Use this command in a local area network to change the command execution process. A record or a file may be in use on the network, and several retries may be required to access it. You can request from −1 to 32,000 retries with the <expN> parameter. Use minus 1 from within programs to set up a checking condition with infinite tries. However, this will prevent you from using dBASE IV for as long as your program is executing the infinite retries.

If you set the numeric value to 0, with no ON ERROR command, dBASE IV displays a **Please Wait, another user has locked this record or file** while it tries an infinite number of times to lock the record or file. You can use the **Esc** key to interrupt the program from retrying endlessly if you prefer.

If you specify a number greater than zero, no error message is displayed.

## See Also

FLOCK(), NETWORK(), RLOCK()/LOCK, UNLOCK

# SET SAFETY

SET SAFETY prevents you from unintentionally overwriting an existing file by providing a screen message that asks you to verify that you really want to write over the file.

## Syntax

SET SAFETY ON/off

## Defaults

The default for SET SAFETY is ON.

## Usage

With SET SAFETY ON, if a file with the same name as the one you're creating already exists, dBASE IV displays an error box:

**File already exists**
**Overwrite Cancel**

The selection is on **Overwrite**.

If you want to overwrite the existing file, press ↵. The existing file is then overwritten, and the data it contains is lost. If you don't want to overwrite the existing file, move the selection bar over to **Cancel** and press ↵. Then, give a new name to the file you are creating.

With SET SAFETY OFF, you can overwrite files or destroy data without receiving a warning.

If you ZAP a database file when SET SAFETY is ON, you will be asked to verify that you want to delete this file.

## See Also

COPY, COPY FILE, INDEX, JOIN, SAVE, SORT, TOTAL, UPDATE, ZAP

# SET SCOREBOARD

If SET SCOREBOARD is ON and SET STATUS is OFF, then dBASE IV displays the scoreboard keyboard indicators on line 0 of the screen.

## Syntax

SET SCOREBOARD ON/off

## Default

The default for SET SCOREBOARD is ON.

## Usage

With SET SCOREBOARD ON and SET STATUS OFF, you see scoreboard indicators on line 0 of the screen. These include **Del** to indicate that a record is marked for deletion, **Ins** for insert mode, **Caps** for the caps lock mode, and **Num** for the numeric pad lock indicator.

With SET STATUS ON, when you enter a value that is outside a specified range, the correct RANGE is displayed on the message line. If you turn STATUS OFF while the message is displayed, the message moves to the scoreboard area on line 0 of the screen.

If SET STATUS is on, then these indicators are displayed on the status bar, and SET SCOREBOARD has no effect.

With both SET SCOREBOARD OFF and SET STATUS OFF, dBASE IV does not display these indicators.

## See Also

SET MESSAGE, SET STATUS

# SET SEPARATOR

SET SEPARATOR changes the symbol used for the numeric separator from the comma (,), which is the U.S. convention, to a specified character.

## Syntax

SET SEPARATOR TO [ < expC > ]

## Default

The default for SET SEPARATOR is the comma.

## Usage

The separator character defined as < expC > can be only one character long. The default may be changed in the Config.db file. See Chapter 6, "Customizing dBASE IV."

## Examples

To display a number with period separators, and comma decimal mark:

```
. SET SEPARATOR TO "."
. SET POINT TO ","
. value = "100000.55"
. ? value PICTURE "999,999.99"
100.000,55
```

## See Also

SET POINT

# SET SKIP

SET SKIP supports multiple detail record handling in database files linked with the SET RELATION command. It allows you to access all records in the linked files that match a particular key.

## Syntax

```
SET SKIP TO [ <alias> [, <alias> ]...]
```

## Usage

This command is effective only if a SET RELATION command has been used to link related database files. It allows the record pointer in specified related database files to be updated before the active database file record pointer is changed. This lets you access multiple occurrences of the same key in linked files.

When you update database files linked in a relationship chain, the commands SET RELATION and SET SKIP determine the sequence of the updates to the database files.

With a SET SKIP command active, a command that advances the record pointer begins its updates in the database file which is the first in the SKIP chain. The record pointer is updated in each database file, moving up the relationship chain towards the parent file. The active database may be left off the relation chain; it will be updated last even if it is not specified with the SET SKIP command.

SET SKIP affects all commands that have FOR and WHILE clauses.

## Examples

Use SET SKIP to display all matching records in related database files.

```
. SELECT 3
. USE Items ORDER Order_id
Master index: ORDER_ID
. SELECT 2
. USE Transact ORDER Client_id
Master index: CLIENT_ID
. SET RELATION TO Order_id INTO Items
. SELECT 1
. USE Client          && Master database file.
. SET RELATION TO Client_id INTO Transact
. SET SKIP TO Stock, Client, Transact
. LIST Client, B->Order_id, Stock->Part_name
Record#    Client                  B->Order_id  Stock->Part_name
      1    WRIGHT & SONS, LTD
      2    BAILEY & BAILEY         87-109       TABLE, END
      2    BAILEY & BAILEY         87-109       LAMP, FLOOR
      2    BAILEY & BAILEY         87-112       CHAIR, SIDE
      3    L. G. BLUM & ASSOCIATES 87-106       SOFA, 8-FOOT
      3    L. G. BLUM & ASSOCIATES 87-108       CHAIR, DESK
      3    L. G. BLUM & ASSOCIATES 87-115       LAMP, FLOOR
      4    SAWYER LONGFELLOWS      87-111       CHAIR, DESK
      5    SMITH ASSOCIATES        87-113       BOOK CASE
      5    SMITH ASSOCIATES        87-105       SOFA, 6-FOOT
      5    SMITH ASSOCIATES        87-105       SOFA, 6-FOOT
      6    TIMMONS & CASEY, LTD    87-107       CHAIR, DESK
      6    TIMMONS & CASEY, LTD    87-110       FILE CABINET, 2 DRAWER
      6    TIMMONS & CASEY, LTD    87-110       FILE CABINET, 4 DRAWER
      7    VOLTAGE IMPORTS         87-114       LAMP, FLOOR
      8    PUBLIC EVENTS           87-105       SOFA, 6-FOOT
      8    PUBLIC EVENTS           87-105       SOFA, 6-FOOT
      8    PUBLIC EVENTS           87-116       DESK, EXECUTIVE 5-FOOT
      8    PUBLIC EVENTS           87-116       FILE CABINET, 2 DRAWER
      8    PUBLIC EVENTS           87-116       CHAIR, DESK
```

The Order_id and Part_name fields for the first record do not appear because there is no order for Wright & Sons, LTD. To list only those Clients that have orders, use the command:

```
. LIST Client, B->Order_id, Stock->Part_name FOR "" <> TRIM(B->Order_id
```

## See Also

SET RELATION

# SET SPACE

SET SPACE lets the ? and ?? commands print a space between the printed expressions.

## Syntax

SET SPACE ON/off

## Default

The default for SET SPACE is ON.

## Usage

This command introduces an extra space between fields in the ? or the ?? commands. Use commas between the printed expressions you want to separate with SET SPACE.

## Example

```
. name = "Susan"
Susan
. city = "Boston"
Boston
. ? name, city
Susan Boston
. SET SPACE OFF
. ? name, city
SusanBoston
```

# SET SQL

SET SQL activates the interactive SQL mode in dBASE IV. Once SQL is activated, only SQL commands and a subset of dBASE IV commands are allowed.

## Syntax

SET SQL on/OFF

## Default

The default for SET SQL is OFF.

## Usage

SET SQL ON can be issued from the dot prompt. The dot prompt changes to the SQL prompt, and only valid SQL statements and the allowed subset of dBASE IV commands can be used.

SET SQL OFF can be issued while in SQL mode from the SQL prompt, and the prompt changes to the default dot prompt. When SQL is OFF, only valid dBASE IV commands and functions are allowed.

You cannot use SQL ON/OFF in a program.

If you wish to start in SQL as the default, see Chapter 6, "Customizing dBASE IV," to change your Config.db file.

# SET STATUS

SET STATUS determines whether the status bar displays at the bottom of the screen when you are working from the dot prompt, from within a program, and in full-screen editing commands such as APPEND, EDIT, and READ.

## Syntax

SET STATUS ON/off

## Default

The program default for SET STATUS is OFF. However, STATUS is set ON by the Config.db file which is loaded during installation of single-user dBASE IV. If the Config.db file is not used, the SET STATUS defaults to OFF.

## Usage

When SET STATUS is ON, the status bar displays the command name, the file in use, and the record number out of the total number of records. When STATUS is OFF, this information is not displayed.

If both STATUS and SCOREBOARD are ON, scoreboard information about the keyboard modes is displayed on the status bar. When SET STATUS is OFF, the scoreboard information displays at the top right of the screen.

When you issue commands that cause information on the status bar to change, the status bar is updated.

## Tips

SET STATUS ON can be used with SET FORMAT and READ. However, be sure not to specify fields or text below line 21 (or line 41 in EGA43 mode). If you use the area of the screen below line 21, your programs or data will overwrite the status bar and the message line.

## See Also

SET MESSAGE, SET SCOREBOARD

# SET STEP

SET STEP halts the execution of a program after each instruction. It allows you to step through a program one line at a time. This command is primarily a debugging tool.

## Syntax

SET STEP on/OFF

## Defaults

The default for SET STEP is OFF.

## Usage

During operations with SET STEP ON, a single program instruction is executed at a time. The result of each operation is displayed. Before the next instruction is executed, the following message appears:

**Press SPACE to Step, S to Suspend, or Esc to Cancel...**

Program processing pauses until you enter one of the choices.

## See Also

COMPILE, DEBUG, DO, SET DEBUG, SET DEVELOPMENT, SET ECHO, SET TALK

# SET TALK

SET TALK determines whether the response to certain dBASE IV commands is displayed.

## Syntax

SET TALK ON/off

## Default

The default for SET TALK is ON.

## Usage

During interactive use, SET TALK should be ON. It displays the following information as each record or command is processed: record numbers, memory variables, and the results of commands such as APPEND FROM, COPY, PACK, STORE, and SUM.

SET TALK OFF is used in programs to control what appears on the screen and on the printer.

When SET TALK is OFF, compiler warning messages are not displayed.

## Example

After you SET TALK OFF, a command response is not displayed.

```
. Mnum=12345
12345.00
. SET TALK OFF
. Mchar ="Hello"
```

## See Also

SET DEBUG, SET ECHO, SET STEP

# SET TITLE

SET TITLE turns the catalog file title prompt on and off.

## Syntax

SET TITLE ON/off

## Default

The default for SET TITLE is ON.

## Usage

When SET CATALOG is ON, files are automatically added to the catalog whenever they are created, used, or saved. If SET TITLE is ON, you are prompted for a catalog file title at this time, provided the file isn't already in the catalog. With SET TITLE OFF, this prompt does not appear.

In **Report Design**, **Label Design**, **Database Design**, **Query Design**, **Form Design**, **Program Design**, and **Application Design**, you can add or modify the title (also called the Description), using the pull-down menus.

For other commands, to add a title to a file's catalog record, you must use commands such as EDIT or REPLACE to modify the title field in the catalog file open in work area 10. You can open and modify a catalog file in any work area; 10 is the work area that is opened with SET CATALOG.

## See Also

EDIT, REPLACE, SET CATALOG

# SET TRAP

SET TRAP activates the debugger when an error occurs in a program or **Esc** is pressed.

## Syntax

SET TRAP on/OFF

## Default

The default for SET TRAP is OFF.

## Usage

When SET TRAP is ON, the debugger is called when an error occurs, or if you press the **Esc** key. The program is halted at the line where the error has occurred. If ON ERROR is in effect, it takes precedence over SET TRAP and the ON ERROR statement controls the program.

When SET TRAP is OFF, and no ON ERROR condition is specified, then error conditions give you three options:

- Cancel the program
- Ignore the error
- Suspend the program

If SET TRAP is ON, dBASE IV calls the debugger.

## See Also

DEBUG, ON ERROR, SET DEBUG, SET DEVELOPMENT

# SET TYPEAHEAD

SET TYPEAHEAD specifies the size of the type-ahead buffer.

## Syntax

SET TYPEAHEAD TO <expN>

## Defaults

The default number of characters that the type-ahead buffer holds is 20. The allowable range is an integer between 0 and 32,000.

## Usage

Fast typists can use this command to increase the capacity of the type-ahead buffer so that they do not get ahead of the program and lose keystrokes. SET TYPEAHEAD works only when SET ESCAPE is ON.

In full-screen EDIT or APPEND operations, the type-ahead buffer will hold only 20 characters regardless of a previous setting with SET TYPEAHEAD TO.

## Tips

If you execute an error-handling program using ON ERROR DO <program>, it is a good idea to completely disable the type-ahead buffer. Include SET TYPEAHEAD TO 0 as the first line of this program. This deactivates both the ON KEY command and the INKEY() function. Because an error is an unanticipated condition, disabling the type-ahead buffer is a safety precaution.

## Special Cases

If you try to enter more than the specified number of characters into the type-ahead buffer, all the extra characters are lost. If SET BELL is ON, the beep will sound. If this happens repeatedly, you should increase the size of the type-ahead buffer.

## See Also

INKEY(), ON, SET BELL, SET ESCAPE

# SET UNIQUE

SET UNIQUE determines whether all records with the same key value are included in the index (.ndx) or the multiple index (.mdx) file.

## Syntax

SET UNIQUE on/OFF

## Defaults

The default for SET UNIQUE is OFF.

## Usage

If you create a new index file while SET UNIQUE is ON, and several records have the same key value, only the first record that dBASE IV encounters with that value is included in the new index file.

Whenever you REINDEX an index file that was created with UNIQUE, the file retains its UNIQUE status, regardless of whether SET UNIQUE is ON or OFF.

Adding new records or editing existing records in the database file when a unique index is open does not change the unique status of the index file. This means that if you APPEND a record that contains an index key that is already in the index file, that record does not become a part of the index file, although it gets added to the database file.

Similarly, EDITing a record and changing its key to one which is already in the unique index file removes that record from the index file.

## Programming Note

To restore an index file to a non-unique status, rebuild the index with SET UNIQUE OFF and the INDEX ON command.

When a key field in a UNIQUE index is changed and this record contains hidden duplicate records, the duplicate records are not brought forth until you do a REINDEX.

## Example

To LIST all of the Part_names in the Items database file without repeating a part:

```
. USE Stock
. SET UNIQUE ON
. INDEX ON Part_name TO Itemname
100% indexed                            10 Records indexed
. LIST Part_name

Record#    Part_name
     12    BOOK CASE
      4    CHAIR, DESK
     11    CHAIR, SIDE
     15    DESK, EXECUTIVE 5-FOOT
      8    FILE CABINET, 2 DRAWER
      9    FILE CABINET, 4 DRAWER
      7    LAMP, FLOOR
      1    SOFA, 6-FOOT
      3    SOFA, 8-FOOT
      6    TABLE, END
```

## See Also

FIND, INDEX, REINDEX, SEEK, SEEK(), SET INDEX, SET ORDER, USE

# SET VIEW

SET VIEW performs a query (.qbo or .qbe) created in the Control Center or from the dot prompt, or updates the environment contained in a dBASE III PLUS view (.vue) file.

## Syntax

SET VIEW TO <query filename> /?

## Usage

This command performs a dBASE IV query defined in a .qbo or .qbe file. The query may have been created by calling the query designer from the Control Center, or with CREATE/MODIFY QUERY or CREATE/MODIFY VIEW.

You may either enter a view filename as part of the syntax, or use the catalog query clause, ?, to open a menu of all .qbo, .qbe, and .vue files in the open catalog. If there is no catalog open, a list of the files from the current directory is displayed.

SET VIEW TO updates a catalog if one is open and SET CATALOG is ON. If a catalog is in use and SET CATALOG is ON, each file opened will be added to the catalog unless it is already there, and you will be prompted for a title if SET TITLE is ON.

### File Search Order

When you use the SET VIEW TO command, dBASE IV searches first for a .qbo file. If dBASE IV cannot locate a .qbo file in the currently defined paths, it looks for a .qbe file; that is, a query program file that has not been compiled. If it finds a .qbe file, dBASE IV compiles it to a .qbo and executes it.

If you use a .vue file from dBASE III PLUS, dBASE IV will create a .qbe file from it. When you USE this .qbe file, dBASE IV compiles it to a .qbo file and uses this updated view.

dBASE IV view (.qbo or .qbe) files cannot be used in dBASE III PLUS.

### dBASE IV View Files

A dBASE IV view consists of:

- All open database files, index files, and the work area number of each
- All relations between the database files
- The currently selected work area number
- The active field list
- The active filter
- The open format (.fmt) file, if any

## See Also

CREATE/MODIFY VIEW/QUERY, SET CATALOG, SET FIELDS, SET FILTER, SET FORMAT, SET INDEX, SET ORDER, SET RELATION, SET TITLE

# SET WINDOW

SET WINDOW sets a default window that allows you to edit a memo field within this default window while you are in APPEND, BROWSE, CHANGE, EDIT, or READ.

## Syntax

SET WINDOW OF MEMO TO <window name>

The <window name> is the name of a previously DEFINEd window.

## Usage

This command allows you to use a previously defined window to edit memo fields while you are using a full-screen command such as APPEND, BROWSE, CHANGE, EDIT, or READ.

The WINDOW clause that you specify with the @...GET command for editing memo fields overrides the window specified by the SET WINDOW command.

## Example

The first and the third @...GET commands use the full-screen default window to edit memo fields. The second @...GET uses the window specified with the SET WINDOW OF MEMO command.

```
. DEFINE WINDOW JOHN FROM 5,5 TO 15,60
. DEFINE WINDOW MARY FROM 1,30 TO 11,70
. SET WINDOW OF MEMO TO JOHN
. @ 1,1 GET MEMO1
. @ 2,1 GET MEMO2 WINDOW MARY
. @ 3,1 GET MEMO3
. READ
```

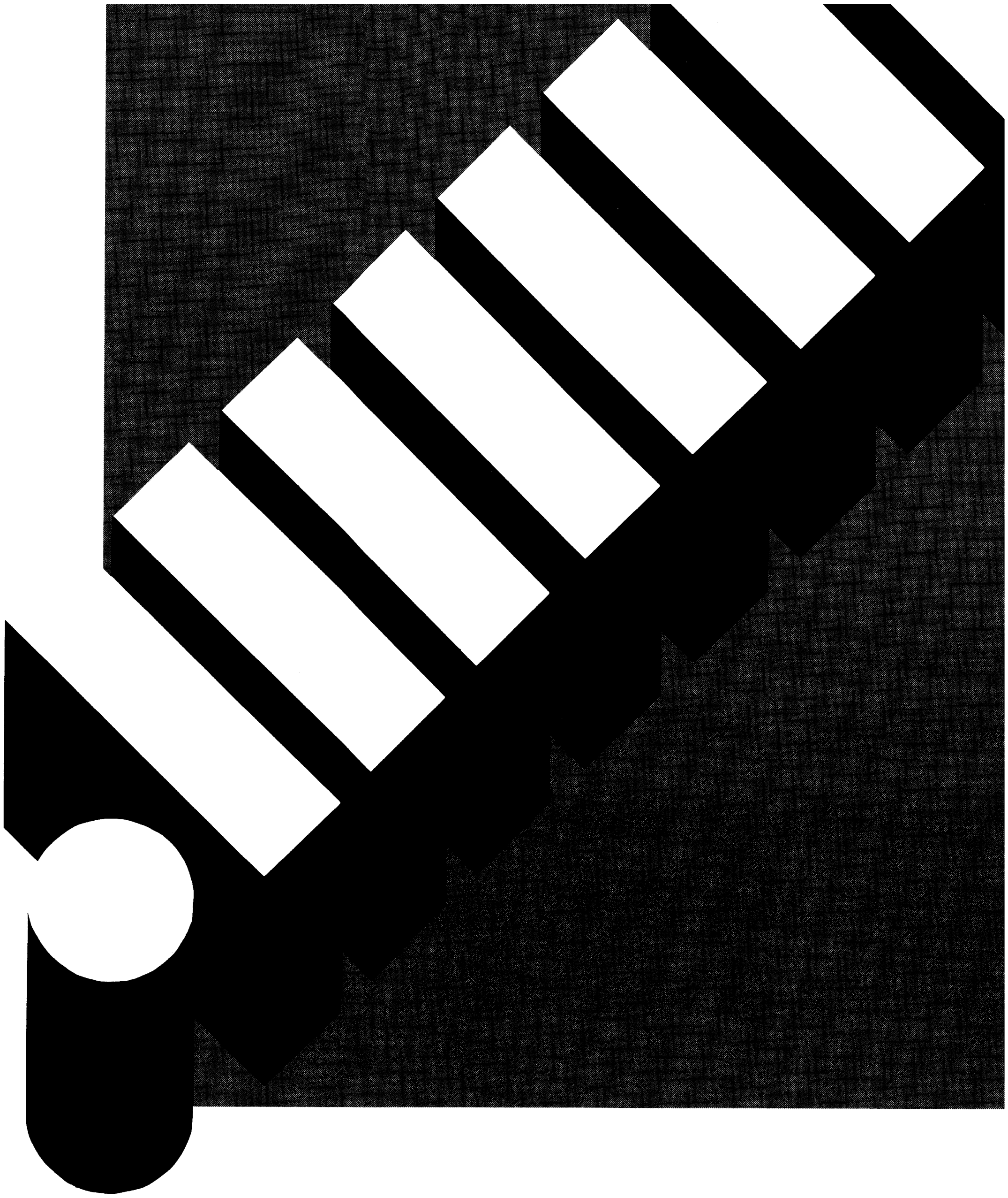
If you try to SET WINDOW OF MEMO TO to an undefined window name, the error message **Window has not been defined** appears.

## See Also

ACTIVATE SCREEN, ACTIVATE WINDOW, CLEAR WINDOW, DEFINE WINDOW, MOVE WINDOW

Language Reference

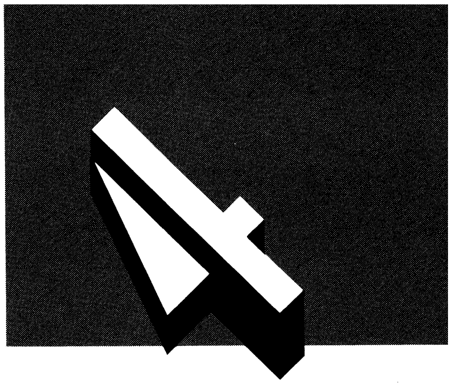
# Functions



i 1 2 3 **4** 5 6 A B C

D E F G In

# Functions



## &

& is the macro substitution function. It substitutes the contents of a memory variable for a literal variable name where dBASE IV would take the literal variable name. This function can be used only for character variables.

### Syntax

& <character variable> [.]

### Usage

This function is used to obtain the contents of a character memory variable when dBASE IV expects a literal value rather than a character expression. Some examples of when dBASE IV expects a literal value are FIND, USE, and other commands that use a filename.

Use the period (.) as the optional macro terminator. When a macro is used as a prefix to a literal string, a macro terminator clearly delimits the end of the macro.

### Examples

Use the FIND command with a memory variable:

```
. Mname = "Peters"
Peters
. USE Client INDEX Cus_name
. FIND &Mname.
. ? RECNO(), Lastname
               4 Peters
```

Substitute a memory variable for a database filename, then USE the variable instead of the filename:

```
. ACCEPT "Enter the database file directory: " TO Mdirectory
Enter the database file directory: \dBASE
. ACCEPT "Enter a database filename: " TO filename
Enter a database filename: Invoices
. USE C:&Mdirectory.\&filename.
. ? DBF()
C:\DBASE\INVOICES.DBF
```

A new method in dBASE IV uses filenames rather than macro substitution. This method does not call the compiler and provides fast execution:

```
. USE "C:"+Mdirectory+:"\"+filename
```

# ABS()

The ABS() function returns the absolute value of a numeric expression without regard to its sign.

## Syntax

ABS( < expN > )

## Usage

Use this function to find the absolute value of a numeric expression. The absolute value of both + 3 and − 3 equals 3. This function enables you to find the difference between two numbers without regard to their sign.

The returned value is always a positive number.

## Example

To determine the number of days between two dates:

```
. date1 = {12/25/88}
12/25/88
. date2 = {04/01/88}
04/01/88
. ? ABS(date1 - date2)
         268.00
. ? ABS(date2 - date1)
         268.00
```

# ACCESS()

The ACCESS() function returns the access level of the current user.

## Syntax

ACCESS()

## Usage

The ACCESS() function allows you to build security into a network application program. The access level returned by this function can be used to test privileges assigned with PROTECT.

## Programming Notes

Use this function to change access levels. For example:

```
IF ACCESS() < 3
```

can easily be changed to:

```
IF ACCESS() < 8
```

To prevent a user from modifying his or her access level, use a compiled program that checks the ACCESS() level. Remove the source file (.prg) from the user's access level. The compiled program files cannot be modified by the user.

The following program example tests the user's access level. If the access level is less than 3, it runs a sub-program. If higher than or equal to 3, it sounds a beep and displays the **Unauthorized ACCESS level** message.

```
ACCEPT "Enter choice " TO Choice
DO CASE
   CASE Choice="1"
      IF ACCESS() < 3
         DO Program
      ELSE
         ? CHR(7)
         ? "ACCESS NOT ALLOWED"
         WAIT
         RETURN TO MASTER
      ENDIF
   CASE Choice="2"
      *
      *
      *
ENDCASE
```

## Special Cases

If dBASE IV does not find the Dbsystem.db file during startup, it does not present the log-in screen to the user. ACCESS() returns a zero if the user has not entered dBASE IV through the log-in screen. A user with an access level of zero cannot access encrypted files. To prevent ACCESS() from returning a zero, it is recommended that you keep the Dbsystem.db file in the same directory as dBASE IV, and always enter through the log-in screen.

If you write programs that use encrypted files, check the user's access level early in the program. If ACCESS() returns a zero, your program might prompt the user to log in again, or to contact the Network Administrator for assistance.

In a single-user environment, ACCESS() always returns a zero.

## See Also

FLOCK(), NETWORK(), PROTECT, RLOCK(), UNLOCK(), USER()

# ACOS()

The ACOS() arccosine function calculates and returns the angle size in radians for any given cosine value.

## Syntax

ACOS( < expN > )

## Usage

The variable ( < expN > ) is a numeric expression that is the cosine of a particular angle. The value of the numeric expression must be between − 1.0 and + 1.0 inclusive.

The response is always a number that represents an angle size in radians between zero and π. The SET DECIMALS and SET PRECISION commands determine the number of decimal points and the accuracy displayed.

## Example

To assign the angle in radians represented by a cosine value, determined in a four-level substitution, to a memory variable:

```
. SET DECIMALS TO 4
. x = 8.4852
          8.4852
. y = 12
          12.0000
. Mcos = x / y
          0.7071
. angle = ACOS(Mcos)
          0.7854
```

## See Also

ASIN(), ATAN(), ATN2(), COS(), DTOR(), FIXED(), FLOAT(), PI(), RTOD(), SET DECIMALS, SET PRECISION, SIN(), TAN()

# ALIAS()

The ALIAS() function returns the alias name of a specified work area. It returns the name of the current work area if you do not specify a work area.

## Syntax

ALIAS([ < expN > ])

## Usage

dBASE IV allows you to open the same database file in nine different work areas, provided you specify a unique alias each time you USE the same database and use the AGAIN keyword.

The ALIAS() function returns the filename or unique alias associated with any work area. The optional variable < expN > specifies which work area to check and is any valid numeric expression between 1 and 10. If you do not specify a work area, the current work area is assumed.

If you use a decimal number such as 7.5 for < expN > , it is truncated to 7.

# ASC()

The ASC() function returns the ASCII decimal code of the first character from a character expression.

## Syntax

ASC( < expC > )

## Usage

Appendix G lists the decimal ASCII values for the IBM character set 0 through 255. Please refer to this appendix to identify a decimal value returned by this function.

## Examples

```
. ? ASC("Nestle")
          78.
```

```
. Number = "123"
         123
```

```
. ? ASC(number)
          49.
```

# ASIN()

The ASIN() arcsine function calculates and returns the angle size in radians for any given sine value.

## Syntax

ASIN( < expN > )

## Usage

The variable ( < expN > ) is a numeric expression that is the sine of a particular angle. The value of the numeric expression must be between − 1.0 and + 1.0 inclusive.

The value returned is always a floating point number that represents an angle size in radians between $-\pi/2$ and $+\pi/2$. The SET DECIMALS and SET PRECISION commands determine the number of decimal points and the accuracy displayed.

## Examples

To find the angle in radians represented by a sine value:

```
. ? ASIN(.5000)
            .5236
```

To assign the sine value to an expression:

```
. star = .5000
            0.5000
. X = ASIN(star)
            0.5236
```

## See Also

ACOS(), ATAN(), ATN2(), COS(), DTOR(), RTOD(), SET DECIMALS, SET PRECISION, SIN(), TAN()

# AT()

The AT() function returns a number that shows the starting position of a character string within a second string.

## Syntax

AT( < expC > , < expC > / < memofield name > )

## Usage

The contained character string is called a substring. If the substring is not contained within the second expression, the function returns a zero.

## Example

Using the Client database file, check the Lastname field for the name "Wright":

```
. USE Client
. ? AT("Wright", Lastname) < > 0
.T.
. SKIP
CLIENT: Record No 2
. ? AT("Wright", Lastname) < > 0
.F.
```

## See Also

LEFT(), RIGHT(), SUBSTR()

# ATAN()

The ATAN() arctangent function calculates and returns the angle size in radians for any given tangent value.

## Syntax

ATAN( < expN > )

## Usage

The variable ( < expN > ) is a numeric expression that is the tangent of a particular angle. The range is between $+\pi/2$ and $-\pi/2$.

The returned value is always a number that represents an angle size in radians. The SET DECIMALS and SET PRECISION commands determine the number of decimal points and the accuracy displayed.

## Examples

To find the angle in radians represented by a tangent value:

```
. ? ATAN(1.000)
          0.7854
```

To assign the tangent value to an expression:

```
. Star = 1.000
. mvar = ATAN(star)
           0.7854
```

## See Also

ACOS(), ASIN(), ATN2(), COS(), DTOR(), RTOD(), SET DECIMALS, SET PRECISION, SIN(), TAN()

# ATN2()

The ATN2() arctangent function calculates and returns the angle size in radians when the cosine and sine of a given point are specified.

## Syntax

ATN2( < expN1 > , < expN2 > )

< expN1 > is the sine of a particular angle, and < expN2 > is the cosine of that same angle. The value of the expression < expN1 > / < expN2 > must fall within the range of + π and − π.

## Usage

This function returns values in all four quadrants, and is equivalent to ATAN(x/y). It is easier to use than ATAN(x/y) because it eliminates divide-by-zero errors.

The returned value is always a number that represents an angle size in radians between + π and − π. The SET DECIMALS command determines the accuracy of the display.

## Example

This example shows an integrated usage of trigonometric functions. You get the sine and cosine of 30 degrees with the degrees to radians conversion function DTOR(), while concurrently saving those values to the memory variables x and y. Next, you use the ATN2() function inside the radians to degrees conversion function RTOD() to get the degrees that correspond to this sine cosine pair. The reason for using the memory variables is to take advantage of the 20-place internal numeric accuracy of dBASE IV without typing large numeric values or changing the decimals setting.

```
. x=SIN(DTOR(30))
   .50

. y=COS(DTOR(30))
   .86

. ? RTOD(ATN2(x,y))
           30.00
```

## See Also

ATAN(), COS(), DTOR(), RTOD(), SET DECIMALS, SIN(), TAN()

# BAR()

The BAR() function returns the BAR number of the most recently selected BAR from a pop-up menu.

## Syntax

BAR()

## Usage

Use this function to get the BAR number of the last selected BAR from the currently active pop-up menu.

The BAR() function returns a zero if:

- There is no active pop-up menu
- No pop-up menu has been defined
- The **Esc** key was pressed to DEACTIVATE the active pop-up menu

The BAR() function returns the BAR number (such as 1, 2) if the pop-up menu was defined using the DEFINE BAR command.

The BAR() function returns the line number for the prompt, counting the first line within the pop-up screen area as 1, if the pop-up prompts were defined using the DEFINE POPUP command.

## Example

In a PROCEDURE in a program file, use BAR() to evaluate and act on a pop-up menu selection.

```
PROCEDURE Viewproc
DO CASE
   CASE BAR() = =
      DO Add_rec
   CASE BAR() = 2
      DO Edit_rec
   CASE BAR() = 3
      DO Del_rec
   CASE BAR() = 4
      DEACTIVATE POPUP
ENDCASE
RETURN
```

## See Also

ACTIVATE POPUP, DEFINE POPUP, POPUP(), PROMPT()

# BOF()

The BOF() function indicates the beginning of the file.

## Syntax

BOF([ < alias > ])

The optional alias can either be a numeric expression from 1 to 10, or a character expression from A through J.

## Usage

This function is intended for programming applications in which the database is read in reverse order. A logical true (.T.) is returned when an attempt is made to move the record pointer before the first logical record of the specified file.

## Special Case

If no database is in USE, BOF() returns a logical false (.F.).

## Example

In a procedure that edits records and moves the record pointer according to the exit key value, use BOF() as a test to avoid moving the record pointer to a nonexistent record.

```
DO WHILE .T.
   EDIT NEXT 1
   DO CASE
      CASE STR(READKEY(),3) $ " 6, 262"
         * ---PgUp key was pressed.
         SKIP -1
         * ---Do not try to EDIT records before BOF().
         IF BOF()
            GO TOP
         ENDIF
      *
      *
   ENDCASE
ENDDO
```

## See Also

EOF()

The CALL() function is used in programming to execute binary program modules.

## Syntax

CALL( < expC > , < expC > / < memvar name > )

where < expC > is a binary filename without its extension.

## Usage

This function is used with the LOAD command to execute binary programs written in other languages (such as assembly or C) within dBASE IV programs. First, LOAD a subroutine, then execute it using the CALL() function. You can then use any value returned by the function in a dBASE IV expression.

When the function is executed, the address of the first argument is loaded into the data segment (DS) and BX register. Program control is transferred to the beginning of the specified loaded module for execution.

Memory variables and array elements may be passed to the binary program module. To pass memory variables to the CALL() function, you must first define them. To use arrays with CALL(), you must declare them and reference their element coordinates. You can pass up to seven parameters to the called module.

## Example

This program example LOADs the binary file into memory. The file Getdrive.asm is on one of the Samples Disks. Next, the array Mdrives is DECLAREd. The first element of Mdrives is assigned a single space, the remaining nine elements are null. The first print statement (?) prints a message for the user about default drive status, and assigns the default drive to the first element of Mdrives. The second print statement verifies that the first element of Mdrives is, in fact, C.

```
. LOAD Getdrive
. DECLARE Mdrives[10]
. Mdrives[1] = " "
. ? "The current drive is", CALL("Getdrive",Mdrives[1])
The current drive is C
. ? Mdrives[1]
C
```

## See Also

DECLARE, LOAD, STORE

# CDOW()

The CDOW() function returns the name of the day of the week from a date expression.

## Syntax

CDOW( < expD > )

## Usage

The date expression is a memory variable, a field, or any function that returns date type data.

## Examples

```
. ? CDOW({02/29/88})
Monday
```

To determine if the current date is a weekday or a weekend:

```
. ? "Today is " + IIF(CDOW(DATE()) = "S", "on a weekend.", "a weekday.")
Today is a weekday.
```

## See Also

CTOD(), DATE(), DAY(), DOW(), DTOC()

# CEILING( )

The CEILING() function calculates and returns the smallest integer that is greater than or equal to the value specified in the numeric expression.

## Syntax

CEILING( < expN > )

## Usage

Use this function to find the smallest integer that is greater than or equal to a given value. The value returned is the same data type as the specified numeric expression.

## Examples

```
. first = 123
        123.00
. second = 10
         10.00
. ? CEILING(first / second)
         13.00
```

CEILING(), unlike ROUND(), always returns an integer closer to zero.

```
. ? CEILING(-5.556)
         -5.00
. ? ROUND(-5.556,0)
         -6.00
```

## See Also

FLOOR(), ROUND()

# CHANGE()

The CHANGE() function determines if a record has been changed since it was opened in a multi-user environment.

## Syntax

CHANGE()

## Usage

This function is valid only for records that have been updated with the CONVERT command.

The CONVERT command creates a new database file with the extension .cvt from the database file that is in USE. You must put this new file with the .cvt extension in USE for the CHANGE() function to work.

The new .cvt database file contains a hidden field called "Count" inside another hidden field called _DBASELOCK. The "Count" field contains a two-digit hexadecimal number that the CHANGE() function reads.

The CHANGE() function checks the *count* for the fields in the current record to see if the count has been updated since the record was opened. It returns a logical true (.T.) if the count value of the current record differs from the value stored in the field _DBASELOCK.

It returns a logical false (.F.) if the values are the same.

**NOTE**
*This function is not totally reliable if used during transaction processing. When a record is updated in a transaction, its count is immediately updated. If you use the CHANGE() function at this point, it will return a true. However, if the transaction is not completed and a rollback occurs, you will not know that the change to the record is no longer valid.*

## See Also

CONVERT, FLOCK(), LKSYS(), RLOCK(), SET REFRESH

# CHR()

The CHR() function converts a numeric expression to a character.

## Syntax

CHR( <expN> )

## Usage

This function enables you to send any value from the IBM extended character set to a printer or the screen. <expN> must be an integer from 1 to 255.

Note that not all printers support the IBM extended character set. Check the ASCII code table for your printer if a character from the extended set that you can display will not print.

You can also use CHR() to send control codes to the printer.

## Examples

To display capital A, which is equal to ASCII decimal 65:

```
. ? CHR(65)
A
```

Use ASCII decimal 7 to sound the bell and display a screen message:

```
. ? CHR(7)+"Be more careful"
Be more careful
```

When you use CHR() in comparisons, CHR(0) *must* be on the left side of the equation. When dBASE IV performs character string comparisons, it reads what is on the right side first. Because CHR(0) is a null string, if it is on the right side, dBASE IV reads no further and returns a true (.T.).

```
. mem = "abc"
abc
. ? mem = CHR(0)
.T.
. ? CHR(0) = mem
.F.
```

## See Also

???, ASC()

# CMONTH()

The CMONTH() function returns the name of the month from a date expression.

## Syntax

CMONTH( < expD > )

## Usage

The date expression is a memory variable, a field, or any function that returns date type data.

## Examples

If the system date is 05/15/88:

```
. ? CMONTH(DATE())
May
```

To determine which month is 76 days from the current date, first store the name of the month to a memory variable:

```
. Mmonth = CMONTH(DATE()+76)
July

. ? "We will visit you in " + Mmonth
We will visit you in July
```

## See Also

DATE(), MONTH()

# COL()

The COL() function returns the current column position of the cursor.

## Syntax

COL()

## Usage

The COL() function may be used for relative screen addressing. Use it to control cursor positioning from within a program. Although the special operator $ can also be used with @...SAY and @...GET commands to indicate the cursor position on the display or the printed page, COL() can do it easier and faster.

For example:

@ 1, COL() + 5 is the same as @ 1, $ + 5.

Both statements move the screen cursor five columns to the right of the current position.

## Examples

Use COL() within a program to find the current cursor position on the screen. For example, if you want to end a loop when the cursor is at column 70, part of the program might include:

```
DO WHILE COL()< 70
   @ 5,COL() + 5 SAY 'X'
ENDDO
```

To print the sentence "This is relative addressing" starting on row 7, column 10:

```
@ 7, 10 SAY "This is "
@ 7, COL() SAY "relative addressing"
```

## See Also

@, PCOL(), PROW(), ROW()

# COMPLETED()

The COMPLETED() function determines whether a transaction has completed; that is, a BEGIN TRANSACTION command has been terminated with an END TRANSACTION command, or a successful ROLLBACK has occurred.

## Syntax

COMPLETED()

## Usage

Use this function to monitor transaction processing, from the dot prompt or from within programs. This function is set to true (.T.) by default. When you issue a BEGIN TRANSACTION command, COMPLETED() is changed to false (.F.). When you issue an END TRANSACTION command, it is reset to true (.T.).

## Example

To test for the successful completion of a transaction in a program file:

```
BEGIN TRANSACTION
   .
   .
   .
END TRANSACTION
IF .NOT. COMPLETED()
   ROLLBACK
     IF .NOT. ROLLBACK()
        IF ISMARKED()
             RESET
        ENDIF
        ? "The transaction was not completed and could not"
        ? "successfully be restored."
        RETURN
     ENDIF
ENDIF
```

## See Also

BEGIN/END TRANSACTION, CHANGE(), CONVERT, LKSYS(), ROLLBACK, ROLLBACK()

# COS()

The cosine COS() function calculates and returns the cosine value for any angle size in radians.

## Syntax

COS( < expN > )

## Usage

The variable ( < expN > ) is a numeric expression that is the size of an angle measured in radians. There are no limits on this numeric expression.

The SET DECIMALS command determines the number of decimal points and the accuracy of the display.

## Examples

```
. ?COS(.7854)
            0.7071
```

To assign the cosine value to a memory variable expression:

```
. Mvar =.7071
            0.7071
. Mcos = COS(mvar)
            0.7602
```

## See Also

ACOS(), ASIN(), ATAN(), ATN2(), DTOR(), RTOD(), SET DECIMALS, SIN(), TAN()

# CTOD()

The CTOD() function converts a date that has been entered or stored as a character string to a date type variable.

## Syntax

CTOD( < expC > ) /{expC}

## Defaults

The format of the character string is normally mm/dd/yy, but this format can be changed by SET DATE and SET CENTURY.

## Usage

The character expression used by CTOD() can range from "01/01/0100" to "12/31/9999". A twentieth century date is assumed if you use only two numbers for the year.

You can also use the curly braces {mm/dd/yy} to create a date variable from a literal value.

## Examples

You can use CTOD() to initialize a date memory variable converting a blank date string (" / / ") or a string specifying an exact date (for example, "01/01/80"). You could then use the date variable, for example, with an @...GET command, to enter a new date in the memory variable as it is displayed on your screen.

```
. STORE CTOD("01/01/80") TO testdate
01/01/80
. ? testdate
01/01/80
```

To convert a character variable to a date variable:

```
. STORE "01/01/80" TO testdate
01/01/80
. ? TYPE("testdate")
C
. STORE CTOD(testdate) TO newdate
01/01/80
. ? TYPE("newdate")
D
```

An alternative to CTOD() is enclosing the date argument in curly braces:

```
. leapdate = {02/29/88}
02/29/88
. SET CENTURY ON
. ? leapdate
02/29/1988
```

## See Also

DATE(), DTOC(), DTOS(), SET CENTURY, SET DATE

# DATE()

The DATE() function returns the system date in the form mm/dd/yy unless the format has been changed by SET CENTURY, SET DATE, or SET MARK.

## Syntax

DATE()

## Usage

The system date must be set at the operating system level.

## Examples

To display the system date:

```
. ? DATE()
04/01/88
```

To store the system date to the memory variable Mdate:

```
. STORE DATE() TO Mdate
05/04/88
```

or:

```
. Mdate = DATE()
02/29/88
```

## See Also

SET CENTURY, SET DATE, SET MARK

# DAY()

The DAY() function returns the numeric value of the day of the month from a date expression.

## Syntax

DAY( < expD > )

## Usage

The date expression is a memory variable, a field, or any function that returns date type data.

## Examples

If the system date is 05/15/88:

```
. ? DAY(DATE())
15
```

To store the day from the system date to the memory variable Mday:

```
. STORE DAY(DATE()) TO Mday
          15.00
. ? Mday
          15.00

. ? TYPE("Mday")
N
```

## See Also

CDOW(), DATE(), DOW()

# DBF()

The DBF() function returns the name of the database file in USE in the currently selected work area.

## Syntax

DBF([ < alias > ])

## Defaults

If no database file is open in the specified work area, DBF() returns a null string.

## Usage

This function is useful in programming to enable the program to work with a file that is in USE in another work area.

## Examples

To determine which file is in USE when you are in the interactive mode:

```
. USE Client
. ? DBF()
C:Client.dbf
```

To clear all the work areas, but retain the currently selected database file in a program:

```
. filename = DBF()
. CLOSE ALL
. USE &(filename)
```

## See Also

ALIAS(), FIELD(), KEY(), MDX(), NDX(), TAG()

# DELETED()

The DELETED() function identifies records that are marked for deletion. A logical true (.T.) is returned if the current record in the specified work area is marked for deletion; if it is not, a logical false (.F.) is returned.

## Syntax

DELETED([ < alias > ])

## Special Case

If no database file is in USE, DELETED() returns a logical false (.F.). If the optional alias is not specified, DELETED() operates on the current work area.

## Example

To determine if the current record is marked for deletion:

```
. USE Client
. ? DELETED()
.F.
. DELETE
1 record deleted.
. ? DELETED()
.T.
```

## See Also

PACK, RECALL, SET DELETED

# DIFFERENCE()

The DIFFERENCE () function determines the difference between two literal strings.

## Syntax

DIFFERENCE( < expC > , < expC > )

## Usage

The DIFFERENCE() function converts two literal strings to SOUNDEX() codes and computes the difference between the two expressions represented by < expC > . It returns an integer between 0 and 4. Two closely matched codes return a difference of 4, and two codes that have no letters in common return a code of 0. One common letter in each string returns a 1.

The two expressions evaluated must be character expressions. Defined variable names may be used.

## Example

To find names with similar SOUNDEX() codes:

```
. USE Client
. LIST Firstname
Record#  Firstname
      1  Frank L.
      2  Sandra
      3  Ric
      4  Kimberly
      5  George J.
      6  Gene
      7  Yuri
      8  Riener
. newname = "Kimbrelee"
Kimbrelee
. LIST Firstname FOR DIFFERENCE(Firstname, newname) > 2
Record#  Firstname
      4  Kimberly
      2  Sandra
      8  Reiner
. LIST Firstname FOR DIFFERENCE(Firstname, newname) > 3
Record#  Firstname
      4  Kimberly
```

## See Also

SOUNDEX()

# DISKSPACE()

The DISKSPACE() function returns an integer representing the number of bytes available on the default drive.

## Syntax

DISKSPACE()

## Programming Note

Use DISKSPACE() with RECCOUNT() and RECSIZE() in application programs that automatically back up database files. This function lets you know if there is sufficient space on the disk for the backup file. See the example under RECSIZE().

## Example

```
SET TALK OFF
USE Client
Mfilesize = (RECCOUNT() * RECSIZE()) + 2000
IF DISKSPACE() > Mfilesize
     Copy TO A:Backup
ELSE
    @ 10,16 SAY "There is not enough room to backup the file."
ENDIF
SET TALK ON
```

## See Also

RECCOUNT(), RECSIZE()

# DMY()

The DMY() function converts the date to a Day/Month/Year format from any valid date expression.

## Syntax

DMY( < expD > )

## Usage

This function converts the date to the following format:

DD Month YY

The day is shown without a leading zero as one or two digits. The month is spelled in full, and the year is shown with the two last digits.

If SET CENTURY is ON, then the format is:

DD Month YYYY

The date expression is a memory variable, a field, or any function that returns date type data.

## Example

To display a date in DMY format:

```
. leapdate = {02/29/88}
02/29/88
. ? DMY(leapdate)
29 February 88
```

## See Also

CDOW(), CMONTH(), DATE(), DOW(), MDY(), MONTH(), SET CENTURY, SET DATE, YEAR()

# DOW()

The DOW() function returns a number that represents the day of the week from a date expression, starting with Sunday as day 1.

## Syntax

DOW( < expD > )

## Usage

The date expression is a memory variable, a field, or any function that returns date type data.

## Examples

If the system date is 05/13/88:

```
. ? DOW(DATE())
          6.00
```

To store the day of the week from the system date to the memory variable Dayofweek:

```
. DOW(DATE()) = Dayofweek
          6.00
. ? Dayofweek
          6.00
```

To determine the number of the day of the week for a date 197 days in the future:

```
. ? DOW(DATE() + 197)
          7.00
```

## See Also

CDOW(), DATE(), DAY()

# DTOC()

The DTOC() function converts a date expression to a character string.

## Syntax

DTOC( < expD > )

## Usage

This function is used to store a date as a character type memory variable or to compare a date to a character.

## Examples

To store the system date as a character type memory variable (assuming the system date is 05/13/88):

```
. STORE DTOC(DATE()) TO testdate
05/13/88
. ? TYPE("testdate")
C
```

To connect a date variable to a character string for use as text:

```
. STORE DATE() TO testdate
05/13/88
. ? TYPE("testdate")
D
. STORE "The date is: "+DTOC(testdate) TO text
The date is: 05/13/88
```

## See Also

CTOD(), DATE(), SET CENTURY, SET DATE

# DTOR()

The DTOR() function converts degrees to radians.

## Syntax

DTOR( < expN > )

## Usage

The expression < expN > is the size of the angle measured in degrees. The DTOR() function returns the angle size in radians.

Convert minutes and seconds to decimal fractions of a degree before using this function.

## Examples

```
. ? DTOR(180)
          3.14
```

Convert 60 degrees 30 minutes 15 seconds to radians:

```
. DTOR(60.525)
          1.056
```

Substitute a variable:

```
. Variable1 = 12
         12.00
. Variable2 = DTOR(Variable1)
           .21
```

## See Also

ACOS(), ATAN(), ATN2(), COS(), RTOD(), SET DECIMALS, SIN()

# DTOS()

The DTOS() function converts a date expression to a character string.

## Syntax

DTOS( < expD > )

## Usage

Use this function when you need to index on a date expression concatenated with a character expression. The variable < expD > is a date expression. This function converts the specified date to a character string of the form CCYYMMDD regardless of the SET CENTURY or SET DATE setting.

## Example

```
. USE Transact
. INDEX ON DTOS(Date_trans) + Order_id TO Tr_date
  100% indexed        12 Records indexed
. LIST NEXT 6 Date_trans, Order_id

Record#  Date_trans  Order_id
      1  02/03/87    87-105
      2  02/10/87    87-106
      3  02/12/87    87-107
      4  02/23/87    87-108
      5  03/09/87    87-109
      6  03/09/87    87-110
```

## See Also

CTOD(), DATE(), DTOC(), SET CENTURY, SET DATE

# EOF()

The EOF() function indicates the end of a file. A logical true (.T.) is returned when the last logical record of the specified database file is passed.

## Syntax

EOF([ <alias> ])

## Usage

When EOF() is true, RECNO() is RECCOUNT() + 1. Also, certain commands (such as SKIP) return the error message **End of File encountered.**

## Special Cases

If the optional alias is not specified, EOF() operates on the current work area.

If no database file is in USE in the specified work area, EOF() returns a logical false (.F.).

## Example

To test for the end of a file:

```
. USE Client
. GO BOTTOM
. ? EOF()
F
. SKIP
. ? EOF()
.T.
```

## See Also

BOF(), ERROR(), FOUND(), RECCOUNT(), RECNO()

# ERROR()

The ERROR() function returns the number corresponding to the dBASE IV error message that caused an ON ERROR condition.

## Syntax

ERROR()

## Usage

The ERROR() function returns the error message number of the error trapped by the ON ERROR command. Error messages and their numbers are listed in Appendix A of this manual, where appropriate corrective actions are suggested when possible.

**NOTE**
*For this function to return a number, an ON ERROR command must be active. RETURN and RETRY clear the error number and message that dBASE IV recorded.*

## Example

In a program that prompts the user for the name of a PFS file to import, a routine is needed to trap the error message **Not a valid PFS file**. The value of this ERROR() is 140.

```
* Main.PRG
ON ERROR DO Err_proc
ACCEPT "Enter the name of the PFS file: " TO filename
IMPORT FROM &(filename) TYPE PFS
* PROCEDURE Err_proc
ON ERROR
DO CASE
   CASE ERROR() = 140
      ? "&(filename) is not a PFS file."
      *
      *
      *
ENDCASE
RETURN
```

## See Also

DEBUG, LINENO(), MESSAGE(), ON ERROR, PROGRAM(), RETRY, RETURN

# EXP()

The EXP() function returns the value that results from raising the constant e to the power of <expN>.

## Syntax

EXP( <expN> )

## Usage

Given the equation $y = e^x$, <expN> is the value of x. For any exponent x to the base e, the function returns the value of y from the equation. The returned value is a real number. The SET DECIMALS setting determines the accuracy of the displayed answer.

## Examples

Any number raised to the power of 1 equals the number. Raising e to 1 returns the value of e.

```
. ? EXP(1.000)
          2.72
```

To get the value at the end of a logarithmic calculation:

```
. x = log(25) + log(25)
     6.44

. ? EXP(x)
     625.00
```

## See Also

LOG(), SET DECIMALS, SET PRECISION

# FIELD()

The FIELD() function returns the field name of the specified field number from the file structure of the selected database file.

## Syntax

FIELD( < expN > [, < alias > ])

## Usage

If the optional alias name is not specified, FIELD() operates on the current work area.

The FIELD() function returns all field names in upper-case letters. If the numeric expression is not between 1 and 255, or if the numeric expression refers to an unused field, dBASE IV returns a null string. For example, if you have a file structure with 28 field names, and you ask for any field above the 28th field name, you get a null string.

Because the FIELD() function addresses each field by its number, it enables you to handle fields as an array.

## Example

Use the Client database file.

```
. USE Client
. ? FIELD(3)
LASTNAME
. number = 2
          2.00
. Mfield = FIELD(number)
CLIENT
```

## See Also

DBF(), LUPDATE(), RECCOUNT(), RECSIZE()

# FILE()

The FILE() function verifies the existence on a selected disk of a specified filename and returns logical true (.T.) if the file exists on the disk.

## Syntax

FILE( < expC > )

## Usage

The FILE() function requires both the filename and its extension. If the file is not on the default drive and directory, you must give the complete file specification including the drive and directory name.

The FILE() function is not case sensitive.

## Examples

If the default drive is C and Client.dbf is on drive A, this function does not find it without the drive specification:

```
. ? FILE("Client.dbf")
.F.
```

It does not find the file without the extension:

```
. ? FILE("A:Client")
.F.
```

Give the complete file specification:

```
. ? FILE("A:Client.dbf")
.T.
```

## See Also

SET DEFAULT, SET PATH

# FIXED( )

The FIXED ( ) function converts long, real floating point numbers to binary coded decimal numbers.

## Syntax

FIXED( < expN > )

## Usage

The FIXED( ) function is used to convert type F (IEEE long, real floating point) numbers to type N (binary coded decimal) numbers. Some levels of precision may be lost in the conversion. Both number types support scientific notation.

If an expression uses both numeric types, all numbers are converted to type F.

The range is between $10^{308}$ to $10^{-308}$.

## See Also

FLOAT( )

The FKLABEL() function returns the name assigned to the specified function key.

## Syntax

FKLABEL( < expN > )

## Usage

You may program a total of 28 keys: 9 function keys, and 19 function key combinations with either the **Shift** or the **Ctrl** keys. The full-screen SET command shows a list of the programmable keys. You may program these keys from the menus of the full-screen SET command, from the dot prompt with the SET FUNCTION command, or from the Config.db file. (See Chapter 6, "Customizing dBASE IV.")

The FKLABEL() function returns the key label associated with the function key specified by < expN > . Any numeric expression from 1 to 28 is valid.

The **F1** function key is dedicated to the Help function and is not programmable. FKLABEL() starts counting the function keys beginning with 1 = **F2**.

## Example

To determine the FKLABEL() corresponding to a function key, type the following at the dot prompt:

```
. ?FKLABEL(9)
F10
```

## See Also

FKMAX(), SET, SET FUNCTION

# FKMAX()

The FKMAX() function returns an integer representing the number of programmable function keys on the computer keyboard.

## Syntax

FKMAX()

## Usage

The FKMAX() function is used to determine the maximum number of programmable keys supported by dBASE IV.

dBASE IV uses the **F1** key as HELP, and the **Shift-F10** keys to access the macro menu; therefore, these keys are not programmable. The **F11** and **F12** function keys of 101-key keyboards are also not programmable.

## Example

Find the total number of programmable keys:

```
. ? FKMAX()
28
```

This is the sum of **F2** through **F10** (9 keys), plus **Shift-F1** through **Shift-F9** (9 keys), and **Ctrl-F1** through **Ctrl-F10** (10 keys).

## See Also

FKLABEL(), SET, SET FUNCTION

The FLOAT() function converts binary coded decimal (type N) numbers to long, real, floating point (type F) numbers.

## Syntax

FLOAT( < expN > )

## Usage

The expression < expN > is any valid type N number. This numeric data is converted to type F numbers.

## See Also

FIXED()

# FLOCK()

The FLOCK() function provides explicit database file locking from either the dot prompt or from within dBASE IV programs.

## Syntax

FLOCK([alias])

## Usage

The FLOCK() function prevents multiple users from simultaneously updating the same file. Use the FLOCK() function to lock all the records in a file for operations involving the whole file. Use the RLOCK() or LOCK() function to lock one record in a file for efficient multi-user file access.

This explicit locking feature is in addition to the automatic locking that dBASE IV provides when you use commands that work with the entire database file. See the SET LOCK command in Chapter 3 of this manual for a list of all dBASE IV commands that provide automatic database file locking, and the level of lock each provides.

A locked file cannot be modified by another user until the lock is released. However, FLOCK() lets other users have read-only access to view the locked file.

If you specify the optional alias name with FLOCK(), dBASE IV attempts to lock the file in the designated work area. If you do not specify an alias, dBASE IV locks the file in the current work area. The alias name you specify must be the same name as the ALIAS keyword or the default filename.

If the file lock attempt is successful, dBASE IV opens the file for exclusive use, and FLOCK() returns a logical true (.T.). The file remains locked until you:

- Unlock it with the UNLOCK command
- Close the file
- Quit dBASE IV

If the file is already locked by another user, FLOCK() returns a logical false (.F.). FLOCK() always returns a logical true (.T.) if it is not used on a network.

If you lock a file while it is actively related to other open files, then all the files in the relation are automatically locked. Unlocking any one of the related files automatically unlocks the other files.

## Example

The following program file segment uses the FLOCK() function to lock a database file. If the lock is unsuccessful after 250 attempts, the procedure Time_out is executed:

```
SET REPROCESS TO 250
IF .NOT. FLOCK()
   DO Time_out
   RETURN
ENDIF
```

## See Also

ACCESS(), RLOCK()/LOCK(), SET LOCK, SET REPROCESS, UNLOCK

# FLOOR()

The FLOOR() function calculates and returns the largest integer that is less than or equal to the value of the specified numeric expression.

## Syntax

FLOOR( < expN > )

## Usage

Use this function to find the largest integer that is less than or equal to a given value. The returned value is the same data type as the specified numeric expression < expN > .

## Example

```
. first = 121
         121.00
. ? FLOOR(first / 10)
         12.00
```

## See Also

CEILING(), INT(), ROUND()

# FOUND()

The FOUND() function returns a logical true (.T.) if the previous FIND, LOCATE, SEEK, or CONTINUE command was successful.

## Syntax

FOUND([ < alias > ])

## Usage

FOUND() is used in programming to branch on the result of a search command. There is one FOUND() per work area. If the optional alias is not specified, FOUND() operates on the current work area.

If files are linked by a SET RELATION TO command, dBASE IV searches the related database files as you move in the active file with FIND, LOCATE, SEEK, or CONTINUE commands. FOUND() is set to a logical true (.T.) or a logical false (.F.) both in the active work area and in the related work areas, depending on the result of the search command.

If you move the record pointer with any command other than FIND, LOCATE, SEEK, or CONTINUE, the result of FOUND() is a logical false (.F.).

## Example

To check the results of a search on an unindexed file:

```
. USE Client
. LOCATE FOR Lastname = "Peters"
Record = 4
. ? FOUND()
.T.
. CONTINUE
End of LOCATE scope
. ? FOUND()
.F.
```

## See Also

CONTINUE, EOF(), FIND, LOCATE, LOOKUP(), SEEK, SEEK(), SET NEAR, SET RELATION

# FV()

The FV() function calculates the future value of equal regular deposits into an investment that yields a fixed interest rate for a certain number of time periods.

## Syntax

FV( <payment>, <rate>, <periods> )

## Usage

<Payment> is a numeric expression which can be negative or positive.

<Rate> is interest rate. This is always a positive number. It is assumed that the interest is compounded once each period. If you use a yearly interest rate and compound monthly, the <rate> expression is your rate divided by 12 months.

<Periods> is a numeric expression that represents the number of payments. Each period is the equal time interval between payments.

The output of FV() is a fixed point number representing the total deposits plus the interest generated and compounded.

## Example

```
. INPUT "Enter payment: " TO payment
Enter payment: 1.00
. INPUT "Enter interest rate: " TO rate
Enter interest rate: .02
. INPUT "Enter number of payments: " TO periods
Enter number of payments: 24
. ? FV(payment, rate, periods)
          30.42
```

## See Also

CALCULATE, PV()

# GETENV()

The GETENV() function returns the contents of a DOS environmental system variable such as PATH or COMSPEC. The name of the system variable may be entered as a character string, a character variable, or a combination of both.

## Syntax

GETENV( < expC > )

## Usage

GETENV() enables you to check the settings of the DOS environment commands. If dBASE IV does not find the environment command parameter, it returns a null string.

The variables to an operating system environment are commands such as PROMPT, PATH, and COMSPEC. See your DOS operating system reference manual for a list of the system environment commands. These are usually implemented from the Autoexec.bat file.

You may also need to refer to the SHELL command in the DOS manual if you need to increase the environment space. You may run out of DOS environment space if you use many DOS configuration commands or a long path statement.

## Example

Assume that your default drive is C and you have a path statement with several directories. To find the DOS setting for "PATH":

```
. ? GETENV("path")
C:\;C:\DOS;C:\DBASE...
```

All other directories you may have in your path statement are also displayed.

## See Also

OS()

# IIF()

IIF() stands for *immediate IF* and is a shortcut to the IF...ENDIF programming construction.

## Syntax

IIF( < condition > , < exp1 > , < exp2 > )

## Usage

You can use this function from the dot prompt or inside programs. If the logical expression is true, IIF() returns the result of the first expression; otherwise, it returns the result of the second expression.

< exp1 > and < exp2 > can be character, numeric, logical, or date expressions. Both expressions must be the same type.

You can use the IIF() function from the dot prompt, or with commands such as CREATE/MODIFY REPORT and CREATE/MODIFY LABEL. The IIF() function allows you to replace a field or a value in an expression with one of two values, based on the evaluation of a condition.

## Examples

In the following program, you can replace the first set of commands with the one IIF() command line that immediately follows them. This example uses the IF...ENDIF structure:

```
IF Sex = "F"
   Mname = "Ms. " + Last_name
ELSE
   Mname = "Mr. " + Last_name
ENDIF
```

This is the one-line replacement using IIF():

```
Mname = IIF(Sex = "F", "Ms. ", "Mr. ") + Last_name
```

You could use this IIF() construction to print labels with the appropriate title by including IIF() in the label contents area.

If you have a database file with the date fields Start_date and End_date, you can extend the End_date by 30 days if that date has already passed.

In the interactive mode:

```
. REPLACE ALL End_date WITH IIF(End_date > DATE(), End_date, End_date + 30)
```

If you have already assigned a value to the logical memory variable Is_error, the following routine converts the variable to a number, then saves it in the numeric memory variable Error_code. If Is_error is true, the new memory variable is 1, otherwise it is 0.

```
error_code = IIF(is_error, 1, 0)
```

# INKEY()

The INKEY() function returns an integer representing the most recent key pressed by the operator. It does not halt program execution.

## Syntax

INKEY([n])

## Usage

This function is often used in programs as a condition for branching. It lets you use function and cursor movement keys within a dBASE IV program.

The INKEY() function returns an integer between 0 and 255, corresponding to the decimal ASCII value of the character in the type-ahead buffer. INKEY() returns the control-key equivalent of special keys, such as arrow keys and **Ctrl-Home**. It also returns negative values for combinations of function keys with **Ctrl** and **Shift** keys.

If there are several characters in the type-ahead buffer, INKEY() returns the ASCII value of the first character in the buffer and clears the character from the type-ahead buffer. The optional numeric argument specifies the number of seconds that INKEY() will wait for a key press before it returns control to the calling program.

INKEY() returns a zero if no key is pressed.

Returned decimal codes for all keys and key combinations (except the **Alt** key combinations) are listed in Table 4-1. The values represented by **Ctrl-S** (decimal 19) and **Ctrl-[** (decimal 27 or **Esc**) are not trapped by INKEY().

The **Alt** key combination with the function keys is intercepted by the keyboard macro handler; therefore, these key combinations may not be used with INKEY().

Keyboard codes returned by INKEY() for the **Alt** key combined with the alphanumeric keys are listed in Table 4-2. The **Alt** codes are the same for upper-case and lower-case letters. The values returned are decimal.

Table 4-1 INKEY() returned values

| **Key Names** | | **Decimal value** |
|---|---|---|
| **Ctrl-A** | **Home** | 1 |
| **Ctrl-B** | **Ctrl-→** | 2 |
| **Ctrl-C** | **PgDn** | 3 |
| **Ctrl-D** | → | 4 |
| **Ctrl-E** | ↑ | 5 |
| **Ctrl-F** | **Ctrl-End** | 6 |
| **Ctrl-G** | **Del** | 7 |
| **Ctrl-H** | | 8 |
| **Ctrl-I** | **Tab** | 9 |
| **Ctrl-J** | | 10 |
| **Ctrl-K** | | 11 |
| **Ctrl-L** | | 12 |
| **Ctrl-M** | | 13 |
| **Ctrl-N** | | 14 |
| **Ctrl-O** | | 15 |
| **Ctrl-P** | | 16 |
| **Ctrl-Q** | | 17 |
| **Ctrl-R** | **PgUp** | 18 |
| **Ctrl-S** | ← | 19 |
| **Ctrl-T** | | 20 |
| **Ctrl-U** | | 21 |
| **Ctrl-V** | **Ins** | 22 |
| **Ctrl-W** | **Ctrl-End** | 23 |
| **Ctrl-X** | ↓ | 24 |
| **Ctrl-Y** | | 25 |
| **Ctrl-Z** | | 26 |
| **Esc** | **Ctrl-[** | 27 |

*(continued)*

Table 4-1 INKEY() returned values (*continued*)

| **Key Names** | | **Decimal value** |
|---|---|---|
| **F1** | **Ctrl-** | 28 |
| **Ctrl-]** | **Ctrl-Home** | 29 |
| **Ctrl-^** | **Ctrl-PgDn** | 30 |
| **Ctrl-** | **Ctrl-PgUp** | 31 |
| **Spacebar** | | 32 |
| **Backtab** | | −400 |
| **Ctrl-Backspace** | | −401 |
| **Ctrl-↵** | | −402 |
| **F2** | | −1 |
| **F3** | | −2 |
| **F4** | | −3 |
| **F5** | | −4 |
| **F6** | | −5 |
| **F7** | | −6 |
| **F8** | | −7 |
| **F9** | | −8 |
| **F10** | | −9 |
| **Ctrl-F1** | | −10 |
| **Ctrl-F2** | | −11 |
| **Ctrl-F3** | | −12 |
| **Ctrl-F4** | | −13 |
| **Ctrl-F5** | | −14 |
| **Ctrl-F6** | | −15 |
| **Ctrl-F7** | | −16 |
| **Ctrl-F8** | | −17 |
| **Ctrl-F9** | | −18 |
| **Ctrl-F10** | | −19 |

(*continued*)

Table 4-1 INKEY() returned values (*continued*)

| **Key Names** | **Decimal value** |
|---|---|
| **Shift-F1** | −20 |
| **Shift-F2** | −21 |
| **Shift-F3** | −22 |
| **Shift-F4** | −23 |
| **Shift-F5** | −24 |
| **Shift-F6** | −25 |
| **Shift-F7** | −26 |
| **Shift-F8** | −27 |
| **Shift-F9** | −28 |
| **Shift-F10** | −29 |

Table 4-2 **Alt** key combination INKEY() values

| **With Letters** | | | | **With Numbers** | |
|---|---|---|---|---|---|
| a | −435 | n | −422 | 1 | −451 |
| b | −434 | o | −421 | 2 | −450 |
| c | −433 | p | −420 | 3 | −449 |
| d | −432 | q | −419 | 4 | −448 |
| e | −431 | r | −418 | 5 | −447 |
| f | −430 | s | −417 | 6 | −446 |
| g | −429 | t | −416 | 7 | −445 |
| h | −428 | u | −415 | 8 | −444 |
| i | −427 | v | −414 | 9 | −443 |
| j | −426 | w | −413 | 0 | −452 |
| k | −425 | x | −412 | | |
| l | −424 | y | −411 | | |
| m | −423 | z | −410 | | |

## Example

The following program segment shows branching to two different procedures depending on the key the user presses. The numeric argument lets the program wait three seconds for the user to press a key before returning to program execution.

```
DO WHILE .T.
    i = INKEY(3)
    DO CASE
          CASE CHR(i) $ "QqO"
               RETURN
          CASE CHR(i) $ "Aa1"
               DO <procedure 1>
          CASE CHR(i) $ "Bb2"
               DO <procedure 2>
     ENDCASE
ENDDO
```

## See Also

CHR(), LASTKEY(), ON KEY, READ, READKEY(), SET TYPEAHEAD

# INT()

The INT() function truncates any numeric expression to an integer.

## Syntax

INT( < expN > )

## Usage

You can discard all digits to the right of the decimal point in a numeric expression by using INT().

## Example

To convert the number 10.23 to an integer:

```
. ? INT(10.23)
          10
. STORE 10.23 TO x
          10.23
. STORE INT(10.23) TO x
          10
```

## See Also

CEILING(), FLOOR(), ROUND()

# ISALPHA()

The ISALPHA() function returns a logical true (.T.) if the specified character expression begins with an alphabetical character.

## Syntax

ISALPHA( < expC > )

## Usage

An alphabetical character is any upper- or lower-case letter between a and z.

Use this function to determine whether or not a string begins with an alphabetical character.

## Examples

To test whether a string begins with an alphabetical character:

```
. ? ISALPHA("abc123")
.T.
. ? ISALPHA("ABC123")
.T.
. ? ISALPHA("123abc")
.F.
```

## See Also

ISLOWER(), ISUPPER(), LOWER(), UPPER()

# ISCOLOR()

The ISCOLOR() function returns a logical true (.T.) if a computer is capable of displaying color.

## Syntax

ISCOLOR()

## Usage

The ISCOLOR() function enables developers to design applications for both color or monochrome environments. This function checks the setting of the hardware address for the graphics adapter card and returns a logical true (.T.) if it finds a color graphics adapter.

## Example

To set the color environment from within a program, you could execute the following commands in a dBASE IV program:

```
IF ISCOLOR()
   SET COLOR TO GR/B,W/R,GR
ELSE
   SET COLOR TO W+
ENDIF
```

If ISCOLOR() returns logical true (.T.), this program could continue to define text and highlight colors for a color display. Otherwise, it would set the display for monochrome.

On monochrome monitors that emulate color screens, this function is not reliable.

## See Also

SET COLOR, SET DISPLAY

# ISLOWER()

The ISLOWER() function returns a logical true (.T.) if the specified character string begins with a lower-case alphabetical character.

## Syntax

ISLOWER( < expC > )

## Usage

Use this function to evaluate and manipulate character strings.

ISLOWER() returns a logical false (.F.) when either an upper-case letter or a non-alphabetical character is in the first position of the expression.

## Example

To test whether an expression begins with a lower-case alphabetical character:

```
. ? ISLOWER("abc123")
.T.
. ? ISLOWER("ABC123")
.F.
. ? ISLOWER("12abc")
.F.
```

## See Also

ISALPHA(), ISUPPER(), LOWER(), UPPER()

# ISMARKED()

The ISMARKED() function checks the current work area, or the optional alias, to determine if dBASE IV has placed a marker in the database file header to indicate that the file is in a state of change.

## Syntax

ISMARKED([ < alias > ])

## Usage

This function answers the question of whether a database is in an integral state or in a state of change. If the database file header is marked for change, ISMARKED() returns a logical true (.T.). If the database file is not marked, this function returns a logical false (.F.). When ISMARKED() returns a logical true, this indicates an incomplete transaction is in progress.

ISMARKED() operates on the current work area, unless you specify an optional alias.

## Example

See the example under the COMPLETED() function in this chapter.

## See Also

BEGIN/END TRANSACTION, COMPLETED(), ROLLBACK

# ISUPPER()

The ISUPPER() function returns a logical true (.T.) if the specified character expression begins with an upper-case alphabetical character.

## Syntax

ISUPPER( < expC > )

## Usage

Use this function to evaluate and manipulate character strings.

ISUPPER() returns a logical false (.F.) when either a lower-case letter or a non-alphabetical character is in the first position of the string.

## Example

To test whether a string begins with an upper-case alphabetical character:

```
. ? ISUPPER("ABC123")
.T.
. ? ISUPPER("abc123")
.F.
. ? ISUPPER("12ABC")
.F.
```

## See Also

ISALPHA(), ISLOWER(), LOWER(), UPPER()

# KEY()

The KEY() function returns the key expression for the index file specified by <expN>.

## Syntax

KEY([<.mdx filename>,] <expN> [,<alias>])

## Usage

If you give an .mdx filename to the function, the numeric expression is interpreted in reference to that file. If the specified file does not exist, then this function returns a null string.

If you do not name an .mdx file, then dBASE IV considers all open index files (including .ndx) in the work area to interpret <expN>.

KEY() operates on the current work area unless you use the optional alias parameter to specify a different work area number, an indirect filename reference, or a database file alias name.

## See Also

MDX(), NDX(), ORDER(), TAG()

# LASTKEY()

The LASTKEY() function returns the decimal ASCII value of the last key pressed.

## Syntax

LASTKEY()

## Usage

This function returns the ASCII value of the last key pressed by the operator. It returns the same values as INKEY().

LASTKEY() is used in programming to get the value of the operator's response and to cause a program response (such as displaying a menu or a message, or executing a command or a procedure).

## Example

Use a function key as an alternate key for terminating a full-screen command. To determine what the terminating key was, use LASTKEY() in a program file:

```
SET FUNCTION 5 TO CHR(23)   && CHR(23)=Ctrl-End key value.
DO WHILE .T.
   * ——GET variables
      .
      .
      .
   @ 19,10 SAY "Press F5 to show a list of Items."
   READ
   * ——Check for function key 5 exit.
   IF LASTKEY() = -4
      DO Showitem
      LOOP
   ELSE
      EXIT
   ENDIF
ENDDO
```

## See Also

INKEY(), READKEY()

# LEFT()

The LEFT() function returns a specified number of characters from a character expression or memo field, starting from the first character on the left.

## Syntax

LEFT( < expC > / < memofield name > , < expN > )

## Usage

The LEFT() function lets you retrieve the first part of a character string or a memo field. This is the same as defining the SUBSTR() function with a starting position of one, and the number of characters to extract with < expN > .

The numeric expression defines the number of characters to extract from the character string. If the numeric expression is zero, a null string is returned.

If the numeric expression is greater than the length of the character string, LEFT() returns the entire string.

## Examples

To extract the first three characters from a character string:

```
. ? LEFT("abcdef",3)
abc
```

## See Also

AT(), LTRIM(), RIGHT(), RTRIM(), STUFF(), SUBSTR(), TRIM()

# LEN()

The LEN() function returns a numeric value indicating the number of characters in a specified character expression or memo field.

## Syntax

LEN( < expC > / < memofield name > )

## Usage

Use this function to determine the number of characters in a database field, memory variable, or a memo field. This function returns a zero if the field contains a null string.

## Examples

To find the actual number of characters in a database field, use TRIM() in conjunction with LEN():

```
. USE Client
. GOTO 2
. ? Lastname
Bailey
. ? LEN(Lastname)
        15
. ? LEN(TRIM(Lastname))
         6
```

LEN() also works on memo fields:

```
. ? LEN(Clien_hist)
       366
```

## See Also

TRIM()

# LIKE()

LIKE() is a programming function used for wildcard comparisons. It is also used to support the predicate in the set-oriented SQL language.

## Syntax

LIKE ( <pattern> , <expC> )

## Usage

Use this function to compare the string on the left with the string on the right. The string on the left contains a pattern including wildcard symbols, the string on the right is compared to this pattern. This function returns a logical true (.T.) or false (.F.).

Two wildcard characters are allowed in the pattern string: the asterisk (*) and the question mark (?). The asterisk stands for any number of characters, and the question mark stands for a single character. Both wildcard characters may be used anywhere and more than once in the pattern string.

LIKE("*abc",astring) will return a logical true value if the character variable or field *astring* ends in the string "abc". Similarly, LIKE("*abc*",any string) will return a true value if *any string* contains the "abc" sequence anywhere in it. The function is case sensitive; the wildcard characters may stand for upper-case or lower-case letters. The defined part of the string must match the case of the string you are looking for with the LIKE() function.

## Examples

The LIKE() function lets you use wildcard symbols anywhere in a string, and use them as often as needed.

```
. ? LIKE("abr*","abracadabra")
.T.
```

```
. LIKE("?a*a*","baba au rhum")
.T.
```

```
. LIKE("?a*a*","barbara")
.T.
```

```
. LIKE("?a*a*","bArbara")
.F.
```

## See Also

SOUNDEX()

# LINENO()

LINENO() returns the file relative line number of the line that is about to be executed in a command or procedure file.

## Syntax

LINENO()

## Usage

The LINENO() function enables developers, when in the Debugger, to set breakpoint conditions to stop a program at a specific line.

If you use the LINENO() function in the Breakpoint window of the Debugger, the Debug window becomes active, and you can enter commands at the **Action** prompt.

You can use the LINENO() function with ON ERROR routines to detect the line number where a program error occurs.

If the command or procedure you are executing consists of more than one line, LINENO() returns the first line number regardless of which line you are on.

## See Also

PROGRAM(), SET DEVELOPMENT

# LKSYS()

The LKSYS() function determines the log-in name of the user who has locked a record or a file, and the date and time of the lock.

## Syntax

LKSYS(n)

n = 0 returns the time when the lock was placed.
n = 1 returns the date when the lock was placed.
n = 2 returns the log-in name of the user who locked the record.

## Usage

This function will return a null string, unless the person who has locked the file has used the CONVERT command to create an invisible field of 8 to 24 characters in the active database file.

When you use the CONVERT command, the database you are converting is copied to a .cvt file before the conversion. dBASE IV then creates a new .dbf file with the hidden field _DBASELOCK.

Both the CHANGE() and the LKSYS() functions read the contents of this invisible field called _DBASELOCK to return information. If CONVERT is used with an argument of 8, then the user log-in name will be truncated, and LKSYS() will return a null string. To get the full user log-in name, use an argument of 24 with CONVERT.

Use this function to get information about the status of a locked record or file. The information returned by LKSYS() is valuable for maintaining work flow in a multi-user system.

Whenever you place an explicit lock on a file with FLOCK(), or on a record with RLOCK(), you must use the UNLOCK command to release these files to other users on the network. Also, dBASE IV automatically locks files that are called up with commands that use the entire database file. Commands that place an automatic lock on files are listed in Chapter 3, under SET LOCK.

## See Also

CHANGE(), CONVERT, FLOCK(), RLOCK(), SET LOCK, UNLOCK

# LOG()

The LOG() function returns the natural logarithm of a specified number.

## Syntax

LOG( < expN > )

## Usage

The natural logarithm has a base of e. The LOG() function returns the exponent in the following equation:

$$y = e^x$$

where x is the numeric expression used by the LOG() function. This must be a positive integer for the value of < expN > . LOG() returns the value of y, which is a type F (long, real) number.

## Examples

Determine the natural log of e which equals 2.71828. Any number raised to itself returns a 1.

```
. SET DECIMALS TO 5
. ? LOG(2.71828)
             1.00000
```

Calculate 4 * 2 using natural logs. Get the value from the EXP() function.

```
. SET DECIMALS TO 2
. x = 4
              4.00
. y = 2
              2.00
. ? LOG(x * y)
              2.08
. ? EXP(2.08)
              8.00
```

## See Also

EXP(), LOG10()

# LOG10()

The LOG10() function returns the common log to the base 10 of a specified number.

## Syntax

LOG10( <expN> )

## Usage

The LOG10() function returns the value for y in the equation:

y = LOG10(x)

X is the numeric expression used by the LOG10() function. This must be a positive integer for the value of <expN>. LOG10() returns the value of y, which is a type F (long, real) number.

## Examples

```
. SET DECIMALS TO 4
. ?LOG10(2.0000)
            0.3010
```

To calculate 2*2 using base 10 logs:

```
. y= log10(2)+log10(2)
             0.6021
```

Use either one of the following expressions to get the antilog:

```
. ?10**y
             4
```

or:

```
. ?10^y
             4
```

## See Also

EXP(), LOG()

# LOOKUP()

The LOOKUP() function searches for a record and returns a value from a field in a specified database file.

## Syntax

LOOKUP ( <return field>, <look-for exp>, <look-in field> )

<return field> and <look-for exp> are any valid dBASE IV expressions. You must specify an alias when referring to expressions outside the active work area.

<look-in field> is the name of a field in the database file which you are searching. dBASE IV searches the current work area, unless you precede the look-in field expression by an alias and a pointer.

## Usage

dBASE IV performs a sequential search unless you have an available index which contains the <look-in field> as its key expression. You can optimize the search by having open .mdx tags and .ndx files.

At the end of the search, dBASE IV positions the record pointer in the first record of the active database file where the "look-for expression" matches the value in the look-in field. If no match is found, the record pointer is at the end of the file.

## Example

In this example, the first print statement (?) is used to demonstrate that LOOKUP() moves the record pointer in the database file searched.

To retrieve information from a selected database file not in the current work area:

```
. USE Transact
. USE Client ORDER Client_id IN 2
Master index: CLIENT_ID
. ? Client ->Client_id
A00001
. ? LOOKUP(Client_id,Client ->Client_id)
PUBLIC EVENTS
. SELECT 2
. DISPLAY Client_id, Client
Record# Client_id Client
      8 A10025    PUBLIC EVENTS
```

# LOWER()

The LOWER() function converts upper-case letters to lower-case letters.

## Syntax

LOWER( < expC > )

## Examples

```
. ? LOWER("THIS IS A NICE DAY")
this is a nice day

. STORE "THIS IS A NICE DAY" TO niceday
THIS IS A NICE DAY
. ? niceday
THIS IS A NICE DAY

. ? niceday = "this is a nice day"
.F.

. ? LOWER(niceday)= "this is a nice day"
.T.
```

To convert all but the first character in a character string to lower case:

```
. charstring = "UPPERCASE"
UPPERCASE
. newstring = LEFT(charstring,1) + SUBSTR(LOWER(charstring),2)
Uppercase
```

## See Also

ISALPHA(), ISLOWER(), ISUPPER(), UPPER()

# LTRIM()

The LTRIM() function removes leading blanks from a character string.

## Syntax

LTRIM( < expC > )

## Usage

Use this function to remove leading blanks.

## Example

To remove leading blanks from a character string formed from a number:

```
. ? STR(11.95,8,2)
       11.95
. ? LTRIM(STR(11.95,8,2))
 11.95
```

## See Also

LEFT(), RIGHT(), RTRIM(), STR(), SUBSTR()

# LUPDATE()

The LUPDATE() function returns the date of the last update of the specified database file.

## Syntax

LUPDATE([ < alias > ])

## Usage

This function displays the last date when the active database file was updated. If no database file is in USE, LUPDATE() returns a blank date.

If the optional alias is not specified, LUPDATE() operates in the current work area. Alias can refer to a work area number, a database file alias name, or an indirect file reference that can be interpreted as a database filename.

## Examples

To make sure that receipts are not entered more than once a day, include the following lines in a data entry program:

```
IF LUPDATE() < DATE()
   Do_entry = "?"
   @ 5,1 SAY "Receipts were last entered on " + DTOC(LUPDATE())+;
             " Enter now? (Y/N)" GET Do_entry PICTURE "Y"
   READ
   IF Do_entry = "Y"
      DO <entry procedure>
   ENDIF
ENDIF
```

## See Also

DBF(), DTOC(), FIELD(), MDX(), NDX(), ORDER(), RECCOUNT(), RECSIZE(), TAG()

# MAX()

The MAX() function returns the larger of two numeric, date, or character expressions.

## Syntax

MAX( < expN1 > / < expD1 > , < expN2 > / < expD2 > )

## Usage

The MAX() function compares two numeric, date, or character expressions, returning the larger of the two. In the case of dates, it returns the later of the two.

This function is different from the aggregate MAX function used with the CALCULATE command. See the CALCULATE command for a description of the aggregate MAX function.

## Example

To determine the greater of two values:

```
. first = 12.32
          12.32
. second = 34.12
          34.12
. ? MAX(first, second)
          34.12
```

To determine the greater of two character expressions:

```
. Name = "Jones"
      Jones
. ? MAX(Name,"Smith")
      Smith
```

## See Also

CALCULATE, MIN()

# MDX()

The MDX() function returns the filename for the .mdx file specified by the index order number.

## Syntax

MDX( < expN > [, < alias > ])

## Usage

This function returns the filename for the open active .mdx directory in the .mdx file list for the current work area. < expN > is a numeric expression specifying the .mdx file position in the index file list specified by the SET INDEX TO < index file list > command. It returns a null string, if there is no list of .mdx files or there is no .mdx file associated with the number you specified.

If the optional alias is not specified, MDX() operates in the current work area.

## See Also

KEY(), NDX(), ORDER(), TAG()

# MDY()

The MDY() function converts the date format to Month Day, Year.

## Syntax

MDY( < expD > )

## Usage

The MDY() function converts any date to a Month (full name of month) Day (two digits), Year (two digits) format. If SET CENTURY is ON, four digits are displayed for the year.

This function returns the date as a character expression.

## Example

To display 02/29/88 in MDY() format:

```
. leapdate = {02/29/88}
02/29/88
. ? leapdate
02/29/88
. SET CENTURY ON
. ? MDY(leapdate)
February 29, 1988
```

## See Also

DMY(), SET CENTURY, SET DATE, SET MARK

# MEMLINES()

The MEMLINES() function returns the number of lines contained in a memo field, when the memo field is word wrapped at the current SET MEMOWIDTH value.

## Syntax

MEMLINES( < memo field name > )

## Usage

This function uses the memo width setting established with the SET MEMOWIDTH command, to determine the number of lines in a memo field.

## Example

This example shows the interaction between the SET MEMOWIDTH TO command and the number of lines of text that the MEMLINES() function returns. SET MEMOWIDTH changes the word wrap position; therefore, the number of lines of text changes accordingly.

Assume that there is a memo field called Mary_mem containing the text:

Mary had a little lamb, whose fleece was white as snow.

```
. SET MEMOWIDTH TO 20
. ? Mary_mem
Mary had a little
lamb, whose fleece
was white as snow.
. ? MEMLINES(Mary_mem)
3
. SET MEMOWIDTH TO 30
. ? Mary_mem
Mary had a little lamb, whose
fleece was white as snow.
. ? MEMLINES(Mary_mem)
2
```

## See Also

MLINE(), SET MEMOWIDTH

# MEMORY()

The MEMORY() function returns the amount of RAM presently unused.

## Syntax

MEMORY([0])

## Usage

This function returns the number of 1024-byte increments (kilobytes) of RAM available.

Use this function to check the amount of available memory before running applications with menus, windows, and arrays that require additional memory, or before you use the RUN/! command to execute DOS commands and programs.

The MEMORY() function does not require any parameters; it returns the same value with or without the optional 0 argument.

## Example

This example uses the MEMORY() function to test if there is sufficient memory to load Command.com before issuing the RUN command:

```
IF MEMORY() < 25
  ? "There isn't enough memory to load Command.com."
ELSE
   RUN COPY Acct_rec.dbf A:
ENDIF
```

## See Also

DISKSPACE(), GETENV(), OS()

# MENU()

The MENU() function returns the name of the active menu.

## Syntax

MENU()

## Usage

This function returns the alphanumeric string for the name of the most recent menu that was activated. If there is no active menu, this function returns a null string.

This function is useful for finding out the name of the menu you were using before you branched to HELP with the **F1** key.

## Example

If the name of the menu that was last activated and is still active is Utility, use the MENU() function to confirm this.

```
. ? MENU()
Utility
```

## See Also

ACTIVATE MENU, DEFINE MENU

# MESSAGE( )

The MESSAGE() function returns the message corresponding to the error which caused an ON ERROR condition.

## Syntax

MESSAGE()

## Usage

The error message returned by the MESSAGE() function can be displayed on the screen or stored to a variable.

See Appendix A for a list of all error messages generated by dBASE IV in single-user operation.

Error messages generated by dBASE IV on a local area network are described in the *Networking with dBASE IV* manual.

## See Also

ERROR(), ON ERROR

# MIN()

The MIN function() returns the smaller value of two numeric, date, or character expressions.

## Syntax

MIN( < expN1 > / < expD1 > , < expN2 > / < expD2 > )

## Usage

The MIN() function returns the smallest number specified by the expressions. In the case of dates, it returns the earlier of the two date expressions.

This function is different from the aggregate MIN() function used with the CALCULATE command. See the CALCULATE command for a description of the aggregate MIN() function.

## Examples

To determine which of two memory variables is smaller:

```
. first = 123.54
          123.54
. second = 232.63
          232.63
. ? MIN(first, second)
          123.54
. ? IIF(MIN(first, second) = first, "First" , "Second") + " is smaller."
First is smaller.
```

To determine the smaller of two character expressions:

```
. Name = "Jones"
      Jones
. ? MIN(Name, "Smith")
      Jones
```

## See Also

CALCULATE, MAX()

# MLINE()

The MLINE() function extracts a specified line of text from a memo field in the current record.

## Syntax

MLINE( < memo field name > , < expN > )

## Usage

The nth line of text of the memo field is extracted. This function uses the memo width setting established with the SET MEMOWIDTH command, to determine the text that the specified line contains.

## Example

This example shows the interaction between the SET MEMOWIDTH TO command and the line of text that the MLINE() function displays. SET MEMOWIDTH TO changes the word wrap position; therefore, the line of text displayed with MLINE() changes.

Assume that there is a memo field called Mary_mem containing the text:

Mary had a little lamb, whose fleece was white as snow.

```
. SET MEMOWIDTH TO 15
. ? Mary_mem
Mary had a
little lamb,
whose fleece
was white as
snow.
. ? MLINE(Mary_mem,2)
little lamb,
. SET MEMOWIDTH TO 30
. ? Mary_mem
Mary had a little lamb, whose
fleece was white as snow.
. ?MLINE(Mary_mem,2)
fleece was white as snow.
```

## See Also

MEMLINES(), SET MEMOWIDTH

# MOD()

The MOD() function returns the remainder from a division of two numeric expressions. MOD() is particularly useful for converting units, such as inches to yards where the division often leaves a remainder.

## Syntax

MOD( < expN1 > , < expN2 > )

## Usage

The MOD() function returns a whole number, the *modulus*, which is the remainder of the division of < expN1 > by < expN2 > .

MOD() returns a positive number if < expN2 > is positive and a negative number if < expN2 > is negative.

The modulus formula is:

< expN1 > − FLOOR( < expN1 > / < expN2 > ) * < expN2 >

where FLOOR is a mathematical function that returns the greatest integer less than or equal to its argument.

## Examples

To determine the MOD() of two numbers:

```
  . ? MOD(14,12)
             2
```

```
  . ? MOD(0,32)
             0
```

```
  . ? MOD(1,-3)
            -2
```

## See Also

FLOOR(), INT()

# MONTH( )

The MONTH( ) function returns a number representing the month from a date expression.

## Syntax

MONTH( < expD > )

## Usage

The date expression is a memory variable, a field, or any function that returns date type data.

## Examples

If the system date is 05/15/87:

```
. ? MONTH(DATE())
            5.00
```

Store the month from the system date to the memory variable Mmonth:

```
. STORE MONTH(DATE()) TO Mmonth
            5

. ? Mmonth
            5
```

## See Also

CMONTH(), DAY(), YEAR()

# NDX()

The NDX() function returns the name of an .ndx file.

## Syntax

NDX( < expN > [, < alias > ])

where < expN > is equal to the ORDER of the index or an arbitrary number if no ORDER is specified. The alias can be a work area number or letter, the alias name of the open database file, or an indirect filename reference.

## Usage

The NDX() function allows programs to manipulate open index files without knowing the names of the files.

The numeric expression is the position of the open index files in the index file list of the currently selected work area. If the user has specified an index file list with either SET INDEX TO < index file list >, or USE INDEX < index file list >, then a numeric expression of 1 through n (where n is the number of index files in the list) implies the index in position 1...n of the list.

An argument greater than n in < expN > implies an .mdx tag and cataloged .ndx files. Any arbitrary number greater than n can be used for this purpose. Then the available indexes are all open .ndx files, .mdx files and the open catalog.

The NDX() function works with the database file in the active work area. It uses the number supplied by the user in < expN > to return the name of the appropriate index file.

If there is no index associated with the number, the function returns a null string.

If the ALIAS name does not exist, an error message **ALIAS not found** is returned.

If the optional alias name is not specified, NDX() operates in the current work area.

## See Also

ALIAS(), DBF(), FIELD(), LUPDATE(), MDX(), ORDER(), RECCOUNT(), RECSIZE(), SET INDEX, SET ORDER

# NETWORK()

The NETWORK() function determines whether or not the system is running on a network. It returns either a logical true (.T.) or a logical false (.F.).

## Syntax

NETWORK()

## Usage

This function is used in programming to determine the system environment in which the program is running, and to branch accordingly.

## Example

```
. ? NETWORK()
.F.
```

## See Also

GETENV(), OS()

# ORDER()

The ORDER() function returns the name of the primary order index file or .mdx tag.

## Syntax

ORDER([ < alias > ])

where the option < alias > is the alias name of an open database file.

## Usage

The ORDER() function determines the name of the controlling index for the active database file, or the database file specified with the alias name.

This function returns the root portion of the filename of the .ndx file in upper case. If there is no .ndx file, then it returns the .mdx tagname, also in upper case.

If the optional alias is not specified, ORDER() operates in the current work area.

## Example

```
. USE Client ORDER Client
Master index: Client
. ? ORDER("Client")
CLIENT
. SET ORDER TO TAG Client_id
Master index: CLIENT_ID
. ? ORDER()
CLIENT_ID
```

## See Also

KEY(), MDX(), NDX(), SET ORDER, TAG()

# OS()

The OS() function returns the name of the operating system under which dBASE IV is running.

## Syntax

OS()

## Usage

Use this function to find out the operating system under which dBASE IV is running.

## Example

```
. ?OS()
DOS 3.10
```

## See Also

GETENV(), VERSION()

# PAD()

The PAD() function returns the prompt PAD name of the most recently selected PAD of the active menu.

## Syntax

PAD()

## Usage

When a menu is active, the cursor can be positioned among the options. Each option is located on a prompt PAD. When you select an option, press the ↵ key and that option PAD becomes the most recently selected PAD. The PAD() function returns the name of that PAD.

## Example

If you have a Utility menu that has three PADs called Display, Sort, and Save, and the Utility menu is still active, use the PAD() function to find out which one you selected last:

```
. ? PAD()
Sort
```

## See Also

DEFINE BAR, DEFINE MENU, DEFINE PAD, DEFINE POPUP, MENU(), ON PAD, ON SELECTION PAD, POPUP()

# PAYMENT()

The PAYMENT() function calculates the constant, regular payment required to amortize a loan with constant interest over a given number of payment periods.

## Syntax

PAYMENT( <principal>, <rate>, <periods> )

<principal> is the principal balance of the loan expressed as a numeric expression. It can be positive or negative.

<rate> is the interest rate expressed as a positive decimal number.

<periods> is a positive numeric expression representing the number of payment periods. Any fractional amounts are rounded off to an integer.

## Usage

This function returns the amount of payment to be made each period to pay off the principal and the interest in the specified number of payment periods.

## Example

To determine the payment of a $7,500 three-year loan at 1.5%:

```
. INPUT "Enter the principal balance: " TO principal
Enter the principal balance: 7500.00
. INPUT "Enter interest rate: " TO rate
Enter interest rate: .015
. INPUT "Enter number of payments: " TO periods
Enter number of payments: 36
. ? PAYMENT(principal, rate, periods)
271.14
```

## See Also

CALCULATE, FV(), PV()

# PCOL( )

The PCOL() function returns the current column position on the printer (relative to _ploffset), and helps keep track of printer column positions within programs.

## Syntax

PCOL()

## Usage

The PCOL() function may be used for relative addressing. The special operator $ can be used with @...SAY and @...GET commands to indicate the column and row position on the printed page. For example:

@ 1,PCOL() + 5 is the same as

@ 1, $ + 5.

Both statements move the print head five columns to the right of the current position.

The value returned by PCOL() is relative to the current _ploffset; that is, it is added to the printer offset value.

The PCOL() function works only with the printer. The $ special operator can be used to control output to both the screen and the printer.

## Example

This program segment determines the current printer row:

```
SET DEVICE TO PRINTER
  @ PROW(), PCOL(), SAY "TEST"
  ? PCOL()
  4
SET DEVICE TO SCREEN
RETURN
```

## See Also

@, COL(), PROW(), ROW(), SET MARGIN, _pcolno, _ploffset

# PI()

The PI() function returns the irrational number 3.14159 which is an approximation of the constant for the ratio between the circumference and the diameter of a circle.

## Syntax

PI()

## Usage

The constant π is used in mathematical and engineering calculations. SET DECIMALS and SET PRECISION parameters determine the accuracy of the displayed number.

Increase your SET DECIMALS value to display more decimal places.

## Examples

```
. ? PI()
              3.14
```

Calculate the area of a unit circle, that is, a circle with a radius of 1. The area always equals π, and the units of measure are the same as the units used to measure the diameter such as inches, kilometers, or miles.

```
. r = 1
              1.00
. a = PI() * r * r
              3.14
```

# POPUP()

The POPUP() function returns the name of the active pop-up menu.

## Syntax

POPUP()

## Usage

The active pop-up menu is the last popup that was activated which has not been deactivated. If there is no active pop-up menu, this function returns a null string.

## Example

If the active pop-up menu is called POP1, then:

```
. ? POPUP()
    POP1
```

## See Also

ACTIVATE POPUP, BAR(), DEFINE POPUP, MENU(), PAD(), PROMPT()

# PRINTSTATUS()

The PRINTSTATUS() function returns a logical true (.T.) if the print device is ready to accept output.

## Syntax

PRINTSTATUS()

## Usage

PRINTSTATUS() checks the status of the most recently selected print device, regardless of how it was selected. The printer may be selected in any number of ways: SET PRINTER ON, SET DEVICE TO PRINTER, **Ctrl-P**, or with any command that allows a TO PRINTER clause.

## Example

The following code checks the status or the print device before printing a report:

```
Mready = ""
DO WHILE .T.
   @ 23,10 SAY "Please ready the printer for the Employee report."
   @ 24,10 SAY "Press P to print, any other key to cancel. ";
           GET Mready PICTURE "!"
   READ
   IF .NOT. PRINTSTATUS() .AND. Mready = "P"
      LOOP
   ENDIF
   IF Mready = "P"
      REPORT FORM Employee TO PRINT
   ENDIF
   EXIT
ENDDO
```

## See Also

SET DEVICE, SET PRINTER

# PROGRAM()

The PROGRAM() function returns the name of the program or procedure that was executing when an error occurred.

## Syntax

PROGRAM()

## Usage

This function returns the name, as a character string, of the program file or procedure that was being executed by a user.

You can use this function in the Breakpoint or Display window of the Debugger, from within a program file, or from the dot prompt if a program is currently suspended. If you call this function from the dot prompt when there is no suspended program, it returns a null string instead of a program name.

The program name that is returned does not include an extension.

If a procedure was executing, PROGRAM() returns the procedure name, not the procedure filename.

The PROGRAM() function does not require any parameters.

## Example

If the following line is included in a program, it will return the name and the line number of the program or procedure which caused an error:

```
ON ERROR ? "The error occurred in " + PROGRAM() + "AT LINE" + STR(LTRIM(LINENO()))
```

## See Also

DEBUG, LINENO(), RESUME, SET DEVELOPMENT, SUSPEND

# PROMPT()

The PROMPT() function returns the PROMPT of the most recently selected popup or menu option.

## Syntax

PROMPT()

## Usage

The PROMPT() function returns the text string representing the PROMPT of the most recently selected option in either a popup or a menu.

If **Esc** is pressed to exit an active menu, this function returns a null string.

If there is no active popup, this function returns a null string.

If the PROMPT is defined with the DEFINE POPUP command (which includes the PROMPT options of FIELD, FILES, and STRUCTURE), then the PROMPT() function returns different character strings for each PROMPT option:

FIELD — Returns the contents of the field from the database
FILES — Returns the complete filename including path and extension in upper case
STRUCTURE — Returns the field name in upper case

## Example

In a program file, use PROMPT() to inform the user what action was selected from a menu. The prompt is displayed on the message line.

```
DEFINE POPUP Main FROM 5,10 TO 12,30
DEFINE BAR 1 OF Main PROMPT "Edit records"
DEFINE BAR 3 OF Main PROMPT "Add records"
DEFINE BAR 5 OF Main PROMPT "Quit"
   .
   .
   .
ON SELECTION POPUP Main SET MESSAGE TO PROMPT()
SET STATUS ON
ACTIVATE POPUP Main
```

## See Also

BAR(), DEFINE POPUP, ON SELECTION PAD, ON SELECTION POPUP, POPUP()

# PROW()

PROW() is the printer row function. It determines the current row on the printer.

## Syntax

PROW()

## Usage

This function is used in programming for relative printer addressing. It can be used to place a page break before printing a block of text that should not be divided.

PROW() is set to 0 after an eject.

## Example

To print *An example of relative addressing* five lines below the current printer line:

```
. SET DEVICE TO PRINTER
. @ PROW()+5,1 SAY "An example of relative addressing"
. SET DEVICE TO SCREEN
```

## See Also

@, COL(), EJECT, PCOL(), ROW(), SET DEVICE, SET PRINTER, _plineno

# PV()

The PV() function calculates the present value of equal, regular payments invested at a constant interest rate for a given number of payment periods.

## Syntax

PV( <payment> , <rate> , <periods> )

## Usage

The PV() function calculates the present value of monies invested. It is used to determine the amount that must be invested now to produce a known future value.

<payment> is a numeric expression representing a constant regular payment. It can be positive or negative.

<rate> is a positive decimal expression representing the rate of interest. If this is compounded annually, use the annual percent rate as a decimal number. If the compounding period is less than a year, then convert the annual interest rate to the compounding interest rate:

> Interest compounded daily = annual rate/365
> Interest compounded monthly = annual rate/12

<periods> is the nearest whole number of payment periods.

## Example

```
. ? PV(1,.06,5)
           4.21
```

## See Also

FV(), PAYMENT()

# RAND()

The RAND() function generates a random number.

## Syntax

RAND([ < expN > ])

where < expN > is an optional numeric expression used as the seed to generate a new random number. If the expression is a negative number, the seed is taken from the system clock.

## Usage

The RAND() function computes a random number with or without a numeric argument. You can repeat the function without an argument, to get subsequent random numbers in that sequence.

This function returns numbers between 0 and 0.999999 inclusive.

The default seed number is 100001. To reset the seed to the default value, use RAND(100001).

## Example

```
. ? RAND(23)
           0.13
. ? RAND()   &&returns the next random number
```

# READKEY()

The READKEY() function returns an integer for the key pressed to exit from a full-screen command. This integer changes if any data is changed during full-screen commands.

## Syntax

READKEY()

## Usage

The READKEY() function helps a program determine what action to take after you exit from one of the full-screen commands: APPEND, BROWSE, CHANGE, CREATE, EDIT, INSERT, MODIFY, or READ.

Table 4-3 shows the integer value of READKEY() for each method of exiting from a full-screen command. Notice that each keypress can return one of two possible codes. If data is not changed, the code is between decimal 0 and decimal 36. If data is changed, the code number for the key is increased by 256.

Table 4-3 READKEY() coding

| Non-Updated Code Number | Updated Code Number | Key Pressed | Keypress Meaning |
|---|---|---|---|
| 0 | 256 | **Ctrl-S**, ←, **Ctrl-H** | backward one character |
| — | 256 | **Backspace** | backward one character |
| 1 | 257 | **Ctrl-D**, →, **Ctrl-L** | forward one character |
| 4 | 260 | **Ctrl-E**, **Ctrl-K**, ↑ | backward one field |
| 5 | 261 | **Ctrl-J**, **Ctrl-X**, ↓ | forward one field |
| 6 | 262 | **Ctrl-R**, **PgUp** | backward one screen |
| 7 | 263 | **Ctrl-C**, **PgDn** | forward one screen |
| 12 | — | **Ctrl-Q**, \ **Esc** | terminate without save |
| — | 270 | **Ctrl-W**, **Ctrl-End** | terminate with save |
| 15 | 271 | ↵, **Ctrl-M** | RETURN or fill last record |
| 16 | — | ↵, **Ctrl-M** | at the beginning of a record in APPEND |
| 33 | 289 | **Ctrl-Home** | menu display toggle |
| 34 | 290 | **Ctrl-PgUp** | zoom out |
| 35 | 291 | **Ctrl-PgDn** | zoom in |
| 36 | 292 | **F1** | HELP function key |

## Example

To determine if the contents of a memory variable were altered and to REPLACE a field if changes were made, include the following in a program:

```
IF READKEY() >= 256   && Data has changed
   REPLACE Name WITH Mname
ENDIF
```

## See Also

INKEY(), LASTKEY(), ON KEY, READ

# RECCOUNT()

RECCOUNT() returns the number of records in the specified database file.

## Syntax

RECCOUNT([ < alias > ])

## Usage

This function provides a faster alternative to using the command GO BOTTOM, followed by the RECNO() function. It is a quick function for counting the number of records in a file.

If no file is in use, RECCOUNT() returns a zero.

RECCOUNT() includes *all* records, even if SET DELETED is ON or SET FILTER is in effect.

## Programming Tips

Use RECCOUNT() with RECSIZE() and DISKSPACE() in application programs that automatically back up database files. This enables you to check if there is sufficient space on the disk for the backup file. See the example in the RECSIZE() entry.

## Example

To find the number of records in the Client database file:

```
. USE Client
. ? RECCOUNT()
           8
```

## See Also

DBF(), DISKSPACE(), RECSIZE()

# RECNO()

The RECNO() function returns the current record number of the specified file.

## Syntax

RECNO([ < alias > ])

## Usage

The RECNO() function returns the following values:

No records in a database file: RECNO() = 1
No database file is in use: RECNO() = 0

If the record pointer moves past the last record in the file (EOF() is a logical true), RECNO() returns a value which is one more than the number of records in the file.

If the record pointer moves backward to before the first record in the file (BOF() is a logical true), RECNO() is 1.

If the optional alias is not specified, RECNO() operates in the current work area.

## Example

To display the current record number:

```
. USE Client
. ? RECNO()
. GO BOTTOM
CLIENT: Record No 8
. ? RECNO()
. SKIP
Client: Record No 9
```

## See Also

SET STATUS

# RECSIZE()

The RECSIZE() function returns the size of a record in the database file in USE.

## Syntax

RECSIZE([ < alias > ])

## Usage

If the optional alias is not specified, RECSIZE() operates in the current work area. If no file is in use, RECSIZE() returns a zero.

## Programming Notes

Use RECSIZE() with RECCOUNT() and DISKSPACE() in application programs that automatically back up database files. This enables you to check if there is sufficient space on the disk for the backup file. You also must know the header size, which is calculated as follows:

32 * < number of fields > + 35.

## Example

To copy a large database file from a hard disk to floppy disks on drive B, you can use the following routine in a program file:

```
USE Filename
Mheadsiz = (32 * <Number of fields> + 35)
SET DEFAULT TO B  && B: is the destination drive.
SCAN
   WAIT "Insert a new disk in drive B, and press a key..."
   IF DISKSPACE() - Mheadsiz <= 0
      ? "Not enough room on that disk."
      LOOP
   ENDIF
   COPY NEXT INT((DISKSPACE() - Mheadsiz)/RECSIZE()) TO Backup
   IF EOF()
      EXIT   && No more records.
   ENDIF
ENDSCAN
CLOSE DATABASE
SET DEFAULT TO <Original drive>
```

## See Also

DBF(), DISKSPACE(), RECCOUNT()

# REPLICATE()

The REPLICATE() function repeats a character expression a specified number of times.

## Syntax

REPLICATE( <expC> , <expN> )

## Usage

This function creates a character string by repeating the character string in <expC> as many times as specified by the integer in <expN>.

The resulting character string must not exceed 254 characters. Therefore, the numeric expression must be a number less than 254 divided by the number of characters in <expC>.

Use the REPLICATE() function whenever you want to repeat a character.

## Example

Use REPLICATE() to create a bar graph of the Total_bill field in the Transact database file. Because most of the values of Total_bill exceed 254, a weighting factor is introduced to ensure that the numeric argument does not exceed 50.

```
. SET HEADING OFF
. USE Transact
. CALCULATE MAX(Total_bill) TO greatest
12 records
            1850
. weight = INT(greatest / 50 )
            37
. LIST Total_bill, REPLICATE("*", INT(Total_bill / weight))
    1   1850.00 **************************************************
    2   1200.00 ********************************
    3   1250.00 *********************************
    4   1250.00 *********************************
    5    415.00 ***********
    6    175.00 ****
    7   1000.00 ***************************
    8    700.00 ******************
    9    125.00 ***
   10    450.00 ************
   11    165.00 ****
   12   1500.00 ****************************************
```

# RIGHT()

The RIGHT() function returns a specified number of characters from a character expression or a variable, starting from the last character on the right.

## Syntax

RIGHT( < expC > / < variable > , < expN > )

## Usage

The RIGHT() function allows you to retrieve the last part of a character string or a variable. The numeric expression defines the number of characters to extract from the character string or variable.

If the numeric expression is zero or negative, RIGHT() returns an empty string.

If the numeric expression is greater than the length of the character string, RIGHT() returns the entire string.

## Example

To extract the last three characters from a character expression:

```
. ? RIGHT("abcdef",3)
def
```

## See Also

AT(), LEFT(), LTRIM(), RTRIM(), STUFF(), SUBSTR()

# RLOCK()/LOCK()

This function is used to lock multiple records. The two function names are interchangeable.

## Syntax

RLOCK( < [expC list > , < alias > ] / [ < alias > ])

## Usage

The character expression list is a list of record numbers (such as 1,2,5,7,9). The current record in the active database is locked if you do not use a record number list or an alias.

Alias may refer to a work area number, a database file alias name, or an indirect file reference. If you use an alias without a list of record numbers, the current record in that database file is locked.

This function locks all listed records and all other active records related to them. Records are related with the SET RELATION TO command. The maximum number of records that can be locked is 50.

RLOCK() provides a shared lock; multiple users have read-only access to the locked records, but only the user that has placed the lock can modify locked records. Use this function to lock a single record, or a group of related records while allowing other users access to *any unlocked record regardless of its position* in the database file.

If all the records in the list and all other records related to them can be locked, the RLOCK() function returns a logical true (.T.). Otherwise, it returns a logical false (.F.) and none of the records are locked.

A record that is locked with RLOCK() is not unlocked until you issue an UNLOCK command, close the database file, or quit dBASE IV.

## Example

Lock record #3 of the Client database file while the Transact database file is selected:

```
. CLEAR ALL
. USE Client INDEX Cus_id IN 2
. USE Transact
. ? RLOCK(3, "Client")
.T.
```

## See Also

FLOCK(), SET LOCK, SET RELATION, UNLOCK

# ROLLBACK()

This function determines whether or not the last ROLLBACK command was successful.

## Syntax

ROLLBACK()

## Usage

ROLLBACK() returns a logical true (.T.) if the last ROLLBACK command was successful. If the ROLLBACK command was not successful, this function remains set to false (.F.) until there is a successful ROLLBACK, or until you exit dBASE IV.

## Example

See the example under COMPLETED() in this chapter.

## See Also

BEGIN/END TRANSACTION, COMPLETED(), ISMARKED(), ROLLBACK

# ROUND()

The ROUND() function rounds numbers to a specified number of decimal places.

## Syntax

ROUND( <expN1>, <expN2>)

## Usage

<expN1> is the number or numeric expression you want to round.
<expN2> is the number of decimal places you want to retain. If
<expN2> is negative, ROUND() returns a rounded whole number.

## Examples

```
. ? ROUND(14.746321,2)
        14.75

. ? ROUND(17.321111,6)
        17.321000

. ? ROUND(10.7654321,0)
        11
. ? ROUND (14911,-3)
     15000
```

Negative numbers round as if they were positive:

```
. ? ROUND(-5.8,0)
  -6

. ? ROUND(-5.2,0)
  -5
```

## See Also

CEILING(), FLOOR(), INT(), STR(), VAL()

# ROW()

The ROW() function returns the row number of the current cursor position.

## Syntax

ROW()

## Usage

ROW() is useful for relative screen addressing. The ROW() value is always a 21 or 24 at the dot prompt, unless a WINDOW is active.

## See Also

@, COL(), PCOL(), PROW(),

# RTOD()

The RTOD() function converts radians to degrees.

## Syntax

RTOD( < expN > )

The expression < expN > is a number representing an angle size in degrees.

## Usage

Use this function to convert radians to degrees.

## Example

Convert 3π/2 radians to degrees:

```
. x = 3 * PI()/2
4.71
. ? RTOD(x)
270.
```

## See Also

ACOS(), ASIN(), ATAN(), ATN2(), COS(), DTOR(), SIN(), TAN()

# RTRIM( )

The RTRIM() function removes all trailing blanks from a character string. This function is identical to the TRIM( ) function.

## Syntax

RTRIM( < expC > ) or TRIM( < expC > )

## Usage

Use this function to trim trailing blanks from fields containing character strings. TRIM( < field name > ) followed by a comma inserts one blank space before the next field. TRIM( < field name > ) followed by a plus sign does not insert any blank space before the next field.

## Examples

The Client database file contains the fields Lastname and Firstname used in this example:

```
. USE Client
. ? Lastname + Firstname
Wright         Fred
```

The first name is separated from the last name by the number of trailing blanks in the Lastname field. To eliminate the blank spaces between the Lastname and Firstname fields:

```
. ? TRIM(Lastname) + Firstname
WrightFred
```

In the following examples, the first and last names are trimmed, but the trailing blanks follow. The field widths remain the same:

```
. DISPLAY TRIM(Lastname), City
Record#  TRIM(Lastname)  City
      1      Wright     New York

. DISPLAY TRIM(Lastname)+", "+Firstname, City
Record#  TRIM(Lastname)+", "+Firstname  City
      1  Wright, Fred              New York
```

## See Also

LEFT(), LTRIM(), RIGHT()

# SEEK()

The SEEK() function performs lookups in indexed database files.

## Syntax

SEEK( < exp > [, < alias > ])

## Usage

The < alias > parameter can refer to the work area number, the database file alias name, or an indirect file reference for an indexed database file that is open, and < exp > is the expression being searched. If no alias is specified, the current work area is used.

The SEEK() function evaluates the specified expression and attempts to find its value in the master index of the database file. It returns a logical true (.T.) if the index key is found, and a logical false (.F.) if it is not found.

## Programming Notes

In a program, SEEK and FOUND() can be combined to execute a specific action when the keyword is found. For example:

```
SEEK M_id
IF FOUND()
   * —— Do something
ENDIF
```

Because SEEK() both moves the record pointer and returns a logical result, SEEK() can be used instead of a FIND/SEEK and FOUND() combination. For example:

```
IF SEEK(M_id)
   * —— Do something
ENDIF
```

## Examples

To move a database file record pointer in an unselected work area:

```
. USE Client INDEX Cus_name
. SELECT 2
. USE Transact
. DISPLAY
      1  A10025 87-105 02/03/87 .T.  1850.00
. ? Client->Client_id
A00001
. ? SEEK(Client_id, "Client")
.T.
. ? Client->Client_id
A10025
```

Use SEEK in a program file to determine if a key field has a matching record in an unselected database file without using the SET RELATION TO command. For example:

```
IF .NOT. SEEK(Client_id, "Client")
   ? "This record does not have a match in the Client database file."
ENDIF
```

## See Also

FIND, FOUND(), LOOKUP(), SEEK

# SELECT()

The SELECT() function returns the number of the highest unused work area.

## Syntax

SELECT()

## Usage

Use SELECT() to determine the number of the highest unused work area. This function returns a number between 1 and 10.

## Example

```
. ? SELECT()
9
```

To open a file in an unselected area:

```
. Filename = "Client"
. USE (Filename) IN SELECT()
```

## See Also

ALIAS()

# SET()

The SET() function returns the status of the SET TO and SET ON commands.

## Syntax

SET( < expC > )

## Usage

The character expression < expC > is the keyword for any valid SET command that is an integer or an ON/OFF value.

The SET() function returns ON, OFF, or an integer. If the SET command controls both ON/OFF and an integer, SET() returns only ON or OFF.

## Examples

To determine the current status of SET CARRY:

```
. ? SET("CARRY")
OFF
```

In a program file, to give the user a chance to change the bell setting:

```
bellstat = IIF(SET("BELL") = "ON", "on.", "off.")
? "Bell is currently set " + bellstat
ACCEPT "Do you wish to change it? (Y/N) " TO change
IF change $ "Yy"
     SET BELL ON
     IF bellstat = "on"
        SET BELL OFF
     ENDIF
ENDIF
```

Use SET() in a program file, to remember the status of the status bar:

```
statflag = SET("STATUS")
   .
   .
   .
SET STATUS &statflag.
```

## See Also

LIST/DISPLAY STATUS, SET, SET STATUS

Chapter 3, "SET Commands"

The SIGN() function returns a number representing the mathematical sign of a numeric expression. It returns a 1 for a positive number, a − 1 for a negative number, and a 0 for zero.

## Syntax

SIGN( < expN > )

where < expN > is a numeric expression.

## Usage

Use this function to determine the sign of a numeric expression without determining the value for the expression.

## Example

Use SIGN() when the result of a calculation must have the same sign as the initial values used, but where the result of the calculation can be of either sign. The following program segment saves the mathematical sign of the base number to a variable M_sign, and uses that variable to determine the sign of the absolute value of the result.

```
FUNCTION Power
PARAMETERS M_base, M_power
* ---Save the sign of the base number.
m_sign = SIGN(M_base)
M_cnt = 1
DO WHILE M_cnt < M_power
   M_base = M_base * M_base
   M_cnt = M_cnt + 1
ENDDO
RETURN(ABS(M_base) * M_sign)
* EOP: Power
```

## See Also

ABS()

# SIN()

The SIN() function returns the trigonometric sine of an angle.

## Syntax

SIN( < expN > )

< expN > is a numeric expression representing the size of the angle in radians.

## Usage

Use this function to get the sine of an angle. No limits are placed on the argument. SIN() returns a type F number.

## Example

The value of sine varies between the limits of + 1 and − 1, passing through zero at 0, π, and 2π radians.

```
. ? SI(PI()/2)
             1
. ? SIN(PI())
             0
. ? SIN(3*PI()/2)
            -1
. ? SIN(2*PI())
             0
```

The SET DECIMALS and SET PRECISION commands determine the accuracy of the display.

## See Also

ACOS(), ASIN(), ATAN(), ATN2(), COS(), DTOR(), PI(), RTOD(), TAN()

# SOUNDEX()

The SOUNDEX() function provides a phonetic match or sound-alike code to find a match when the exact spelling is not known.

## Syntax

SOUNDEX( < expC > )

## Usage

The SOUNDEX() function returns a four-character code by using the following algorithm:

1. It retains the first letter of < expC >, the specified character expression.
2. It drops all occurrences of the letters *a e h i o u w y* in all positions except the first one.
3. It assigns a number to the remaining letters:

| | |
|---|---|
| b f p v | = 1 |
| c g j k q s x z | = 2 |
| d t | = 3 |
| l | = 4 |
| m n | = 5 |
| r | = 6 |

4. If two or more adjacent letters have the same code, it drops all but the first letter.
5. It provides a code of the form "letter digit digit digit". It adds trailing zeros if there are less than three digits. It drops all digits after the third one on the right.
6. It stops at the first non-alpha character.
7. It skips over leading blanks.
8. It returns "0000" if the first non-blank character is non-alpha.

These eight steps produce a four-character code. This code is used as the index to find possible sound-alike matches.

## Example

Use SOUNDEX() to perform lookups on similar names:

```
. USE Client
. INDEX ON SOUNDEX(Firstname) TO Likename
  100% indexed          8 Records indexed
. ACCEPT "Enter a name to lookup: " TO newname
Enter a name to lookup: Kimbrelee
.
. newname = SOUNDEX(Newname)
K516
. SEEK newname
. DISPLAY Firstname
Record#  Firstname
      4  Kimberly
```

## See Also

DIFFERENCE(), LOCATE

The SPACE() function generates a character string consisting of a specified number of blanks.

## Syntax

SPACE( < expN > )

## Usage

The largest number of spaces you can specify with the SPACE() function is 254.

## Example

To create a memory variable consisting of 20 blank spaces, you would type:

```
. STORE SPACE(20) TO blanks
. ? "*" + blanks + "*"
*                    *
```

# SQRT()

The SQRT() function returns the square root of a positive number.

## Syntax

SQRT( < expN > )

## Usage

SQRT() returns a square root value of the number specified in < expN >. The number specified can be either a type N or a type F number, but SQRT() always returns a type F number.

The SET DECIMALS command determines the number of decimal places displayed.

## Example

To determine the square root of 4:

```
. ? SQRT(4)
            2
. ? SQRT(2*2)
            2
. SET DECIMALS TO 3
. STORE 4.000 TO number
            4
. ? SQRT(number)
            2
```

## See Also

SET DECIMALS

# STR()

The STR() function converts a number to a character string.

## Syntax

STR( < expN > [, < length > ] [, < decimal > ])

## Defaults

The default string length is ten characters, and the number is rounded to an integer.

## Usage

The < length > option you specify is the total number of characters in the string created by STR(), including, if applicable, the decimal point, minus sign, and the number of decimal places.

The number you specify for the < decimal > option is the total number of decimal places to output. If you specify a smaller number than there are digits to the left of the decimal in the numeric expression, STR() returns asterisks in place of the number.

If you specify fewer decimals than are present in the numeric expression, STR() rounds the result to the specified number of decimal places.

## Example

To display the number 11.14 * 10 as a character string:

```
. x = 11.14
            11.14
. ? STR(x*10,5)
111
. ? STR(x*10,5,1)
111.4
. ? STR(x*10,5,2)
111.4
```

## See Also

VAL()

# STUFF()

The STUFF() function replaces a portion of a character string with another specified character string.

## Syntax

STUFF( < expC1 > , < expN1 > , < expN2 > , < expC2 > )

## Usage

< exp1 > can be a character expression or a variable name. < expN1 > and < expN2 > are numeric expressions; < expN2 > may be zero or a positive number.

Use the STUFF() function to change part of a character field without reconstructing the entire field. < expC2 > is inserted into the character expression or memo field at the position indicated by < expN1 > . A number of characters indicated by < expN2 > are removed from the right of the string.

If the string starting position indicated by < expN1 > is zero, STUFF() treats it as 0. If it exceeds the length of the variable, it concatenates to the end.

< expN2 > indicates how many characters you want to remove from the original string. If the number of characters is zero, the second character expression is inserted, and no characters are removed from < expC1 > . The new string will not be the same size as the original string if the specified number of characters in < expN2 > differs from the actual number of characters in < expN1 > .

The STUFF() function is not supported in memo fields.

## Example

To remove all occurrences of a hyphen within a field called Field_1 in the entire database file, use a program file:

```
SCAN
   DO WHILE "-" $ Field_1
      REPLACE Field_1 WITH STUFF(Field_1, AT("-", Field_1), 1, " ")
   ENDDO
ENDSCAN
```

## See Also

LEFT(), RIGHT(), SUBSTR()

# SUBSTR()

The SUBSTR() function extracts a specified number of characters from a character expression or a variable.

## Syntax

SUBSTR( < expC > / < memo field name > , < starting position >
  [, < number of characters > ])

## Default

If you omit the number of characters, the function returns a substring that begins with the starting position and ends with the last character of the original character string.

## Usage

If the number of characters you enter is greater than the number of characters between the starting position and the end of the original character expression, the function returns a substring that begins at the specified starting position and ends with the last character of the original character expression. The starting position must be positive.

## Examples

To extract the substring "59" from the string "1958 1959 1960":

```
. ? SUBSTR("1958 1959 1960",8,2)
59
. STORE "English Spanish Italian German French" TO language
English Spanish Italian German French
. ? SUBSTR(Language,9,7)
Spanish
```

## See Also

AT(), LEFT(), LTRIM(), RIGHT(), STR(), STUFF()

# TAG()

This function returns the TAG name in a specified .mdx file.

## Syntax

TAG([ <.mdx filename> ,] <expN> [, <alias> ])

## Usage

TAG() returns a character string representing the name of an index file (.ndx) or .mdx tag for the index specified by <expN> If there is no such index, then the function returns a null string. If no .mdx specification is used, then the <expN> is interpreted with reference to all the open indexes.

If the optional alias name is not specified, TAG() operates in the current work area.

## Example

To display the second TAG of the Client.mdx file:

```
. USE Client
. ? TAG(2)
CLIENT
```

## See Also

DBF(), KEY(), MDX(), NDX(), ORDER()

# TAN()

The TAN() function returns the trigonometric tangent of an angle.

## Syntax

TAN( < expN > )

## Usage

The specified numeric expression is the size of the angle expressed in radians. This trignometric function increases from zero to infinity between 0 to π/2 radians.

This function returns a type F number. The SET DECIMALS and SET PRECISION commands determine the accuracy of the display.

## Example

```
. ?TAN(PI())
          0
```

## See Also

ACOS(), ASIN(), ATAN(), ATN2(), COS(), SIN()

# TIME()

The TIME() function returns the system time as a character string in the format hh:mm:ss.

## Syntax

TIME()

## Usage

To use TIME() in calculations, convert the value returned to a numeric value using SUBSTR() and VAL().

## Examples

```
. ? TIME()
21:22:23
. Mtime = TIME()
21:22:23
. ? Mtime
21:22:23
```

```
* Time calculation program
Start_time = TIME()
Tally = 0
DO WHILE Tally < 100
     Tally = Tally + 1
ENDDO
End_time = TIME ()
Old_time = VAL(SUBSTR(Start_time, 1, 2)) * 3600 +;
             VAL(SUBSTR(Start_time, 4,2)) * 60 +;
             VAL(SUBSTR(Start_time, 7,2))
New_time = VAL(SUBSTR(End_time, 1, 2)) * 3600 +;
             VAL(SUBSTR(End_time, 4,2)) * 60 +;
             VAL(SUBSTR(End_time, 7,2))
Difference = New_time - Old_time
? "Elapsed time was",LTRIM(STR(Difference)), "seconds."
```

## See Also

DATE()

# TRANSFORM( )

The TRANSFORM() function allows PICTURE formatting of character, logical, date, and numeric data without using the @...SAY command.

## Syntax

TRANSFORM( <exp> , <expC> )

## Defaults

The result of the TRANSFORM() function is always character type data, even if the first expression specifies a different type.

## Usage

The character expression <expC> defines the PICTURE format of <exp>, which may specify a character string or a number.

The TRANSFORM() function lets you use a PICTURE format with characters and numbers and to DISPLAY, LABEL, LIST, and REPORT the data according to the PICTURE format.

## Examples

To format the Lastname field, from the Client database file, with spaces or hyphens between the letters:

```
. USE Client
. ? TRANSFORM(Lastname, "@R X X X X X X")
W r i g h t
. ? TRANSFORM(Lastname, "@R X-X-X-X-X-X")
W-r-i-g-h-t
```

To format the Total_bill field, from the Transact database file, with a leading dollar sign and commas or with a following credit symbol:

```
. USE Transact
. ? TRANSFORM(Total_bill, "@$ ##,###.##")
$1,850.00
. ? TRANSFORM(Total_bill, "@C")
 1850.00 CR
```

## See Also

@...PICTURE

# TYPE()

The TYPE() function evaluates a character expression and returns a single upper-case character that indicates whether the expression is character, numeric, logical, memo, date, float, or undefined. It is a test for the existence of a variable or the validity of an expression.

## Syntax

TYPE( < expC > )

## Usage

The TYPE() function always returns an upper-case letter.

The possible TYPE() values are:

Character = C
Numeric = N
Logical = L
Memo = M
Date = D
Float = F
Undefined = U

## Examples

The argument of TYPE() is a character expression. However, a character string memory variable can be used in the argument to reference other variable types.

```
. string = "Invoiced"
Invoiced
. ? TYPE("string")   && string is a character variable.
C
. ? TYPE(string)
U
. USE Transact
. ? TYPE(string)   && Invoiced is a field in Transact.dbf.
L
```

# UPPER()

The UPPER() function converts lower-case letters to upper-case letters.

## Syntax

UPPER( < expC > )

## Examples

```
. ? UPPER("This is a nice day")
THIS IS A NICE DAY
. STORE "This is a nice day" TO niceday
This is a nice day
```

To display this variable in upper case:

```
. ? UPPER(Niceday)
THIS IS A NICE DAY

. ? niceday="THIS IS A NICE DAY"
.F.

. ? UPPER(niceday)="THIS IS A NICE DAY"
.T.
```

To change a character in the account number (Ac) field of a database file to upper case:

```
. LIST AC
Record #  AC
      1  an03
      2  an39
      3  an47
      4  an21
. REPLACE ALL Ac WITH STUFF(Ac, 1, 1, UPPER(SUBSTR(Ac, 1, 1)))
4 records replaced
. LIST AC
RECORD #  AC
      1  An03
      2  An39
      3  An47
      4  An21
```

## See Also

ISALPHA(), ISLOWER(), ISUPPER(), LOWER()

# USER()

The USER() function returns the log-in name of the user currently logged in to a protected system.

## Syntax

USER()

## Usage

This function returns the log-in name used by an operator currently logged in to a system that uses PROTECT to encrypt files. On a system that does not use PROTECT, this function returns a null string.

## See Also

PROTECT

# VAL()

The VAL() function converts numbers that are defined as characters into a numeric expression.

## Syntax

VAL( < expC > )

## Usage

If the specified character expression consists of leading non-numeric characters other than blanks, dBASE IV returns a value of zero.

VAL() displays the number of decimals set by the SET DECIMALS command.

The VAL() function operates from left to right, converting characters to numeric values until a text character is encountered. Leading blanks are ignored if the argument contains both numeric and non-numeric characters. The leading numeric characters will be converted to a numeric value. Trailing blanks are treated as non-numeric characters and, when encountered, terminate the conversion process.

## Examples

The value of a non-numeric character sequence in VAL() is zero:

```
. ? VAL("ABC")
            0

. SET DECIMALS TO 1
. ? VAL("123.45")
          123.5
. STORE "123.45" TO x
123.45
. STORE VAL(x) TO y

. DISPLAY MEMORY
x          pub C "123.45"
   2 variables defined,      17 bytes used
 254 variables available, 5983 bytes available
```

VAL() displays 123.5, but stores 123.45 to memory.

```
. SET DECIMALS TO 0
. ? STR(Y,6,2)
123.45
. ? 0.00 + Y
          123.45
. ? VAL("A=1231")
            0
. ? VAL("123=A1")
          123
```

## See Also

SET DECIMALS, STR()

# VARREAD()

The VARREAD() function is used to create context sensitive help displays.

## Syntax

VARREAD()

## Usage

The VARREAD() function is used with @ and full-screen editing commands, to return the name of a field or a memory variable that is in the process of being edited.

Use this function while a field is being edited. If you perform a READ, the pending @...GETs are cleared as part of the READ command execution.

## Example

The procedure My_help listed below shows how to use VARREAD() with WINDOW commands to create context sensitive help. When the user presses the **F1** key, the procedure My_help executes. My_help first clears the keyboard buffer with INKEY() because ON KEY does not clear the buffer. The **F1** key press, unless cleared, would be read by INKEY(0) as the user input. The procedure would not pause after displaying a help screen to wait for the user's key press.

```
ON KEY F1 DO My_help
@ 5,5 GET Lastname
@ 6,5 GET Firstname
READ

PROCEDURE My_help
null = INKEY()   && Clear the key buffer.
ACTIVATE WINDOW Help_wind
DO CASE
   CASE VARREAD() = "Lastname"
      ? "Help text for Lastname is displayed in the Help window."
   CASE VARREAD() = "Firstname"
      ? "Help text for Firstname is displayed in the Help window."
ENDCASE
   ? INKEY(0)   && Pause the display.
DEACTIVATE WINDOW Help_wind
RETURN
* EOP: My_help
```

## See Also

APPEND, DEFINE WINDOW, EDIT, READ

# VERSION()

The VERSION() function returns the dBASE IV version number in use.

## Syntax

VERSION()

## Usage

The VERSION() function returns the program version number that appears in the dBASE IV sign-on message.

This function is useful in applications that utilize version-specific features.

## Examples

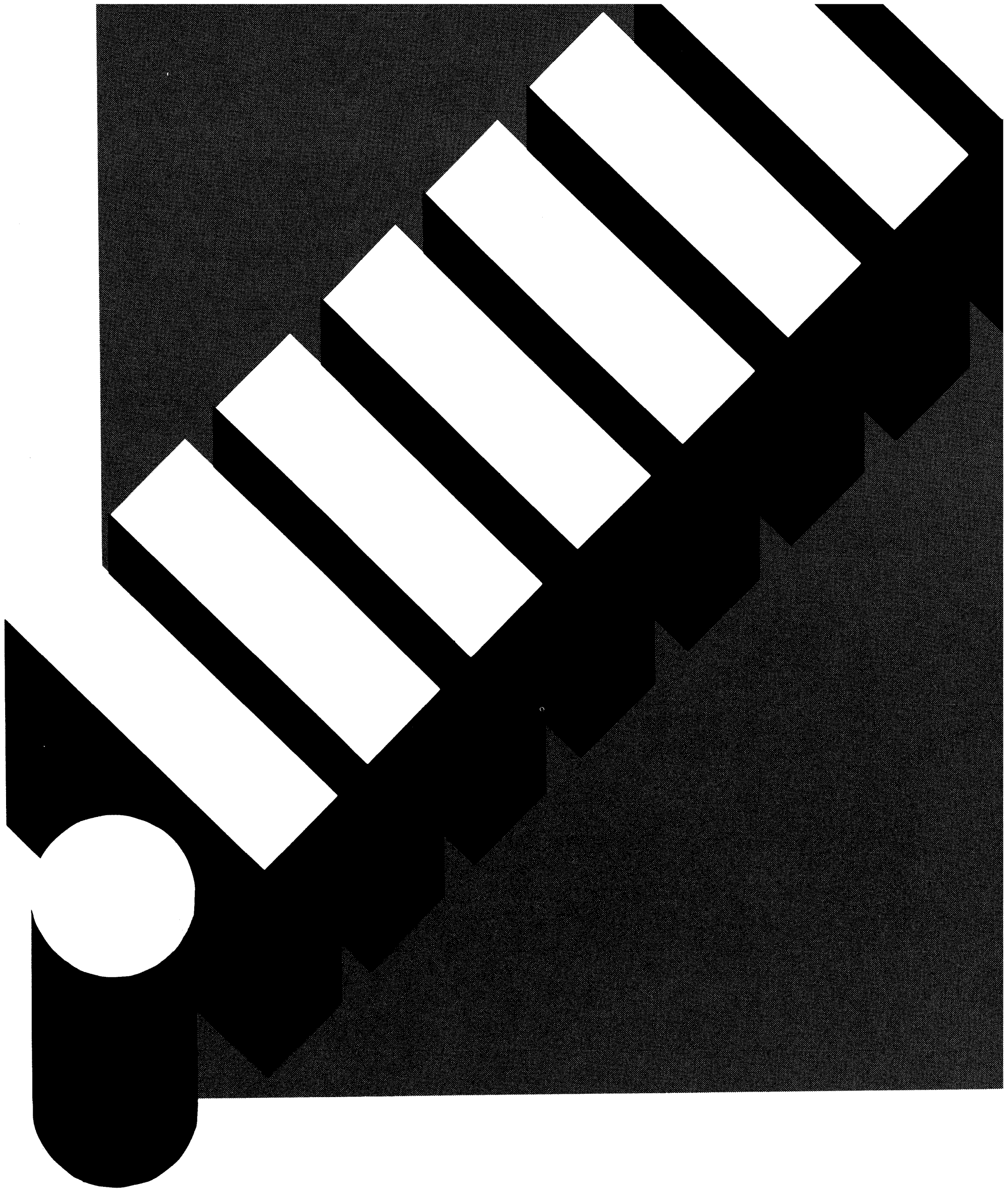
To return the version:

```
. ? VERSION()
dBASE IV 1.0
```

## See Also

OS()

# System Memory Variables



# System Memory Variables

## _alignment

_alignment specifies the alignment of output produced by the ?/?? command with respect to margins when _wrap is true (.T.).

### Syntax

_alignment = "LEFT"/"center"/"right"

### Default

By default, ?/?? output is aligned to the left margin.

### Usage

Use _alignment to control whether output of the ? and ?? commands is left justified, right justified, or centered between the margins. You can change the alignment as often as you want.

This system variable aligns the text *between the margins.* Don't confuse it with the alignment PICTURE functions (B, I, and J), which control the alignment of text *within field templates.*

_alignment has no effect unless the system variable _wrap is set to true (.T.).

### Options

"LEFT" — Text is aligned to the left margin (left-justified).

"CENTER" — Text is centered between the left and right margins.

"RIGHT" — Text is aligned to the right margin (right-justified).

## Example

This procedure prints the date and the page number on the right side of the page. The ?? command causes the page number to print on the same line as the date.

```
_wrap = .T.
SET PRINTER ON
SET TALK OFF
_alignment = "RIGHT"
?? "Page no.", STR(_pageno, 4, 0)," Date:", DATE()
SET PRINTER OFF
_alignment = "LEFT"
```

Depending on the values of DATE() and _pageno, these commands print information similar to the following line:

```
                                        Page no.    1  Date: 07/29/88
```

## See Also

_lmargin, _rmargin, _wrap, @ (PICTURE functions B, I, and J), STR()

# _box

_box controls whether boxes specified with the DEFINE BOX command display.

## Syntax

_box = <condition>

## Default

The default condition is to display boxes, that is, _box is set to true (.T.).

## Usage

Use _box to control whether or not the boxes you defined with the DEFINE BOX command will display. If you set _box to true (.T.), the boxes display. Otherwise, they don't.

You can draw a box with the @...TO command whether _box is true (.T.) or false (.F.).

You can control the exact moment in a program to display a box, or part of a box, with _box. For example, you can interrupt the printing of a box by setting _box to false (.F.) before it has finished printing. You can then later set _box to .T., and the rest of the box will print at that time.

## Example

In the following program, the SPACE() function overwrites 51 characters on the top and bottom lines of the box, so that only the corners print. The assignment statement following SCAN causes the box to print (_box = .T.) for the first and last three records, but not otherwise.

```
*Box.prg
SET TALK OFF
USE Stock ORDER Part_name
DEFINE BOX FROM 10 TO 70 HEIGHT 8 DOUBLE
_box = .T.
?? SPACE(51) AT 15
?
Cnt = 1
SCAN
   _box = (Cnt < 4 .OR. Cnt > (RECCOUNT() - 3))
   ?? Part_name AT 12, Descript
   ?
   Cnt = Cnt + 1
ENDSCAN
?? SPACE(51) AT 15
?
```

The results of the above code look like this:

```
BOOKCASE              WOOD, TEAK, 2-SHELF
CHAIR, DESK           LEATHER, BROWN, HIGHBACK
CHAIR, DESK           LEATHER, BROWN, HIGHBACK
CHAIR, DESK           LEATHER, BROWN
CHAIR, DESK           LEATHER, BROWN
CHAIR, SIDE           PLASTIC, GREY
DESK,EXECUTIVE 5-FOOT WOOD, OAK, FANCY
FILE CABINET,2 DRAWER METAL, BROWN
FILE CABINET,2 DRAWER METAL, BLACK
FILE CABINET,4 DRAWER METAL, BROWN
LAMP, FLOOR           BRASS, 6-FOOT, ENGLISH
LAMP, FLOOR           BRASS, 6-FOOT, ART DECO
LAMP, FLOOR           BRASS, 6-FOOT, ENGLISH
SOFA, 6-FOOT          LEATHER, BROWN, HIGHBACK
SOFA, 6-FOOT          VELVET, GREY, FRENCH
SOFA, 8-FOOT          VELVET, BLUE, FRENCH
TABLE, END            WOOD, OAK, 2-FOOT, SQUARE
```

Figure 5-1 Boxed output

## See Also

DEFINE BOX, SPACE()

# _indent

_indent specifies the indentation of the first line of each new paragraph printed with the ? command when _wrap is true (.T.).

## Syntax

_indent = <expN>

## Default

The default paragraph indentation is 0.

## Usage

The _indent system variable instructs dBASE IV to indent the first line of each new paragraph by the specified number of characters. The value of _indent can range from −(_lmargin), which is the negative value of the _lmargin setting, to (_rmargin − _lmargin − 1). For example, if _lmargin is 10, _indent may be as low as −10, and the first line of each paragraph will output ten columns to the left of the rest of the paragraph. If _rmargin is 75, _indent may be as high as 74.

dBASE IV starts a new paragraph with the specified indentation each time it encounters a ? command. (The ?? command, on the other hand, continues the same paragraph.)

The sum (_indent + _lmargin) must be less than the value of _rmargin. Also, _indent has no effect unless _wrap is set to true (.T.).

Note that the printed output may have additional spacing because of the _ploffset value.

**NOTE**
*Figure 1-1 in Chapter 1, "Essentials," shows a typical _indent on a page layout.*

## See Also

_lmargin, _ploffset, _rmargin, _wrap, ?/??

# _lmargin

_lmargin defines the page left margin for output produced by the ? command when _wrap is true (.T.).

## Syntax

_lmargin = <expN>

## Default

The default left margin is 0.

## Usage

The left margin defines the number of spaces to be output before the unindented text of a paragraph line. You can assign this variable an integer value ranging from 0 to 254. _lmargin has no effect unless _wrap is set to true (.T.).

Note the distinction between _lmargin and _ploffset. _ploffset is the distance from the left edge of the paper to the point where printing starts if the left margin is 0. The left margin is the number of additional columns, if any, where unindented printing *actually* starts. _ploffset only affects printer output.

**NOTE**
*Figure 1-1 in Chapter 1, "Essentials," shows a typical _lmargin on a page layout.*

The sum (_lmargin + _indent) must be less than the value of _rmargin.

## Example

In a program file, use _lmargin to indent a listing relative to its title:

```
SET TALK OFF
USE Client ORDER Client_id
_wrap = .T.
_lmargin = 0
? "Current customer listing"
?
_lmargin = 10
SCAN
   ? Client_id + "   ", Client
ENDSCAN
```

This program generates the following output:

```
Current customer listing

         A00001    WRIGHT & SONS, LTD
         A00005    SMITH ASSOCIATES
         A10025    PUBLIC EVENTS
         B12000    VOLTAGE IMPORTS
         C00001    L. G. BLUM & ASSOCIATES
         C00002    TIMMONS & CASEY, LTD
         L00001    BAILEY & BAILEY
         L00002    SAWYER LONGFELLOWS
```

## See Also

_indent, _ploffset, _rmargin, _wrap, ?/??

# _padvance

_padvance determines how the printer advances the paper.

## Syntax

_padvance = "FORMFEED"/"linefeeds"

## Default

The default for _padvance is "FORMFEED".

## Usage

Use _padvance to control whether dBASE IV advances the paper using line feeds or form feeds.

When using form feeds, dBASE IV feeds the paper according to the printer's internal form length setting.

When using line feeds, dBASE IV determines whether or not the eject was issued while in streaming output mode before it calculates how many line feeds are needed to reach the next page.

If the eject was issued in streaming output mode, dBASE IV uses the formula (_plength − _plineno) to calculate the number of line feeds to send.
This occurs:

- when you issue an EJECT PAGE command without an ON PAGE handler;
- when you issue an EJECT PAGE command with an ON PAGE handler and the current line position is past the ON PAGE line;
- or when the program reaches a PRINTJOB or ENDPRINTJOB and the _peject system memory variable causes an eject.

dBASE IV uses the formula (_plength − MOD(PROW(), _plength)) if the eject is not during streaming output mode. This occurs if you issue an EJECT command, or if you SET DEVICE TO PRINTER and force a page eject with the @ command.

Sending a CHR(12) to the printer always issues a form feed, even if _padvance is set to "LINEFEEDS".

If you're printing short pages, such as checks that are 20 lines long, you don't have to adjust the form length setting on the printer. Instead, set _plength to the length of your output and _padvance to "LINEFEEDS". dBASE IV then bypasses the printer form length setting, since it doesn't send a form feed.

## Options

"FORMFEED" — Advance paper a sheet at a time.

"LINEFEEDS" — Advance paper one line at a time.

## Example

In the following example, the Chk_prin.prg file (which is not provided on disk in your dBASE IV package) prints a check for every record in the Debts.dbf database file:

```
PRIVATE _plength, _padvance
_plength = 20
_padvance = "LINEFEEDS"
USE Debts
SCAN
   DO Chk_prin              && Print a check.
   EJECT                    && Advance to the top of the next check.
ENDSCAN
CLOSE DATABASE
```

## See Also

_peject, _plength, _plineno, EJECT, PROW()

# _pageno

_pageno determines or sets the current page number.

## Syntax

_pageno = <expN>

## Default

The default page number is 1.

## Usage

_pageno works on the data stream that is being output. This *streaming output* is produced by all commands that create output, except the @, @...TO, and EJECT commands.

The _pageno system variable can both determine the current page number and set the page number to a specified value. Use it to print page numbers in a report or, when you are combining documents, to assign a suitable number to the first page of the second document. You can assign _pageno an integer value ranging from 1 through 32,767.

dBASE IV keeps track of the current page number in your streaming output. It increments this variable as it goes to the next page of the streaming output, since the point where the pages break depends upon the actual text in the report. The _pageno value depends upon the value of _plineno and _plength, which are discussed elsewhere in this chapter.

Don't confuse _pageno with _pbpage, the system variable that tells dBASE IV on what page to begin printing.

## Tip

The three system variables _pageno, _pbpage, and _pepage should be consistent. For example, if _pbpage = 3, _pepage = 5, and _pageno = 10, dBASE IV won't print any pages, because the current page number (10) is outside the range of pages to be printed (3 to 5).

The value of _pageno should be less than or equal to the value of _pepage. For the example just given, _pageno should be less than or equal to 5 to print any pages. You can use this feature to print just part of a document by suitably setting _pageno, _pbpage, and _pepage.

## Examples

Suppose you want to combine two documents within dBASE IV. The first document is nine pages long, so you want to start numbering the second document with page ten. _pageno simplifies this task. Since dBASE IV increments _pageno automatically, the page numbers will be continuous from one document to the next.

You can also combine a dBASE IV report with a document created outside dBASE IV. Do this by adjusting the starting page number of the dBASE IV report (assuming it is second in the package). Here is an example:

```
. _pageno = 10

. REPORT FORM Accounts TO PRINT
```

The following footer procedure causes a page number to print centered between hyphens, at the bottom of each page of a report. (Use the ON PAGE command to DO this procedure AT the appropriate line number.)

```
PROCEDURE Footer
PRIVATE _wrap, _alignment
_wrap = .T.
_alignment = "CENTER"
? "-", _pageno PICTURE "@T 9999", "-"
?
RETURN
```

## See Also

_pbpage, _pepage, _plength, _plineno, ON PAGE

# _pbpage

_pbpage specifies the beginning page for a printjob.

## Syntax

_pbpage = <expN>

## Default

The default beginning page number for a printjob is 1.

## Usage

The _pbpage system variable specifies that pages with numbers less than _pbpage are not to be output. dBASE IV will not print any pages numbered below the specified page.

The allowed range of values for this variable is 1 through 32,767. The value of _pbpage must be less than or equal to the value of _pepage (described later in this chapter).

As dBASE IV produces output, it constantly checks that the current page number (_pageno) is greater than or equal to the specified beginning page number (_pbpage). If the current _pageno is less than _pbpage, the output is only scrolled internally to allow _pcolno, _plineno, and _pageno to advance. You will see no output to the screen or printer until _pbpage is reached.

## Tip

The three system variables _pageno, _pbpage, and _pepage should be consistent. For example, if _pbpage = 3, _pepage = 5, and _pageno = 10, dBASE IV won't print any pages, because the current page number (10) is outside the range of pages to be printed (3 to 5).

The value of _pageno should be less than or equal to the value of _pepage. For the example just given, _pageno should be less than or equal to 5 to print any pages. You can use this feature to print just part of a document by suitably setting _pageno, _pbpage, and _pepage.

## Example

Using _pbpage, programmers can let users start program output on a specific page. This option can save users time when a printing failure, such as a paper jam, occurs in the middle of a large printjob. For example, if you have to stop printing a report called Long_rpt.prg, the following will let you restart the print output at the appropriate page number:

```
pbegin = 1
@ 10,20 SAY "Begin on page" GET pbegin PICTURE "99999"
READ
_pbpage = pbegin
DO Long_rpt
```

## See Also

_pageno, _pepage, PRINTJOB/ENDPRINTJOB

# _pcolno

_pcolno positions the subsequent streaming output to begin at a given column of the current line, or returns the current column number.

## Syntax

_pcolno = <expN>

## Usage

_pcolno moves the output position to the desired column on the screen, on the printer, or in the data stream being sent to a file.

*Streaming output* is produced by all commands that create output, except the @, @...TO, and EJECT commands. Streaming output destinations may be controlled by the SET CONSOLE, SET PRINTER, and SET ALTERNATE commands, and by the TO PRINTER/TO FILE <filename> options of commands such as LIST/DISPLAY.

You can assign _pcolno an integer value between 0 and 255.

If _wrap is false (.F.) and SET PRINTER is ON, you can overstrike previously output text by assigning _pcolno a value less than the current value.

If _wrap is true (.T.) and you assign _pcolno a value less than the current value, text in the internal buffer is overwritten and will not be displayed.

Note that the PCOL() function returns the current printhead position of the printer. If SET PRINTER is OFF, the PCOL() value does not change. _pcolno, however, returns or assigns the current position in the streaming output. The value of _pcolno changes as long as data is being output, regardless of the SET PRINTER setting.

Assignment to _pcolno is equivalent to the AT clause of the ?/?? command.

## Example

The following program code segment overstrikes the word "altered" with slashes (/). It does this by backing the printhead to the beginning of the word "altered" and then printing a slash over each character in the word.

```
_wrap = .F.
? "This is some altered"
_pcolno = _pcolno - LEN("altered")
?? REPLICATE("/", LEN("altered"))
?? " changed text."
?
```

This results in the following output:

```
This is some ~~altered~~ changed text.
```

## See Also

_plineno, _rmargin, LEN(), PCOL(), REPLICATE(), SET PRINTER, TRIM()

# _pcopies

_pcopies sets the number of copies to be printed for a printjob.

## Syntax

_pcopies = <expN>

## Default

The default number of copies is 1.

## Usage

The _pcopies system variable determines how many copies a given printjob will print. You can give this variable an integer value from 1 through 32,767.

You can use _pcopies only in programs, because it requires the PRINTJOB/ENDPRINTJOB command. Place the _pcopies variable definition before the PRINTJOB command line.

## Example

This program code allows users to specify the number of copies when printing a report. (See PRINTJOB/ENDPRINTJOB for information on how to code printjobs.)

```
copies = 1
@ 10,20 SAY "Number of copies: " GET copies PICTURE "99"
_pcopies = copies
READ
PRINTJOB
    .          && <Printjob commands>
    .
    .
ENDPRINTJOB
```

## See Also

@, PRINTJOB/ENDPRINTJOB

## _pdriver

_pdriver activates the desired printer driver or returns the name of the current driver.

### Syntax

_pdriver = "<printer driver filename>"

### Default

When you first install dBASE IV, you assign a default printer driver. dBASE IV writes this default driver in your Config.db file. See Chapter 6, "Customizing dBASE IV," for more information on your Config.db printer settings.

You can change the active printer driver by issuing _pdriver = "<printer driver filename>" from the dot prompt.

### Usage

Printer drivers let you print styled text, such as **bold**, <u>underline</u>, and *italics*. The driver you specify interprets all output to the printer except expressions sent with the ??? command.

The driver filename must be a valid DOS filename, and may include a DOS path. You don't need to type the file extension; dBASE IV assumes it to be .pr2.

dBASE IV supports the printer drivers listed in Appendix F. Only one printer driver can be active at a time. Assuming that SET TALK is ON, dBASE IV displays the message **Printer driver installed** if it finds the file, and **File does not exist** if it doesn't.

**NOTE**
*Printer drivers are tailored for each printer to map the IBM extended character set as well as possible. For printers that do not support the IBM extended character set, this means substituting certain characters, such as* –, +, =, *and* :. *If your output does not contain characters from the IBM extended character set, and your printer supports this set, verify that the correct printer driver is active. You can change the current printer driver with the _pdriver system memory variable.*

## Options

The printer drivers you can use with dBASE IV are listed in Appendix F and on the printer selection menu available to you during installation. If your printer is not listed on the menu, select the Generic.pr2 driver.

You may also set _pdriver = "ASCII" to generate ASCII text files, which do not contain printer escape codes.

## Example

This programming example allows you to select the desired printer driver. The @M PICTURE function displays a list of printer types that you can scroll within the window. The DO CASE statement sets _pdriver to the appropriate value when the user chooses a printer type.

```
Mprinter = SPACE(17)
@ 10,20 SAY "Select a printer" GET mprinter;
   PICTURE "@M Epson FX-85,HP LaserJet,IBM QuietWriter,Other";
   MESSAGE "Press Spacebar to view and Return to select."
READ

DO CASE
   CASE mprinter = "Epson FX-85"
      _pdriver = "FX85_1.pr2"
   CASE mprinter = "HP LaserJet"
      _pdriver = "HPLAS100.pr2"
   CASE mprinter = "IBM QuietWriter"
      _pdriver = "IBMQUIET.pr2"
   OTHERWISE
      _pdriver = "GENERIC.pr2"
ENDCASE
```

## See Also

_ppitch, _pquality, ?/??, SET PRINTER

PDRIVER in Chapter 6, "Customizing dBASE IV"

# _pecode

_pecode provides the ending control codes for a printjob.

## Syntax

_pecode = <expC>

## Default

The default _pecode is a null string.

## Usage

The _pecode system variable requires an ENDPRINTJOB to define where the printjob ends, and thus can only be used in programs. Put the variable definition before the ENDPRINTJOB in your program. dBASE IV sends the codes defined by _pecode to the printer when it encounters the ENDPRINTJOB.

Sometimes you want to print a particular report in a different typestyle than you customarily use. For example, you might want to print a wide report in condensed mode. _pscode, described later in this chapter, and _pecode let you start and end the printer control codes that generate the alternate typestyle. The allowed range of codes is 0 through 255, and the length of <expC> is limited to 255 characters.

The ?/?? and ??? commands also send printer control codes to the printer. In general, use ?/?? (STYLE option) to change typestyles on individual items, ??? to change typestyles on a broader basis within a document, and _pecode and _pscode to define the overall typestyle for a printjob.

## Example

This routine sends a printer reset code for a Hewlett-Packard LaserJet at the end of the printjob. (See PRINTJOB/ENDPRINTJOB for information on how to code a printjob.)

```
  _pecode = "{27}{69}"          && Esc E
```

## See Also

_pscode, ?/?? (STYLE option), ???, PRINTJOB/ENDPRINTJOB

Your printer manual has additional information about printer control codes.

# _peject

_peject controls ejecting a sheet of paper before and/or after a printjob.

## Syntax

_peject = "BEFORE"/"after"/"both"/"none"

## Default

The default _peject setting is "BEFORE".

## Usage

You might require a page eject before, after, or both before and after a printjob.

Although you can define it at the dot prompt, you can use _peject only in a program. It requires an identified printjob to be in effect. Define the _peject system variable before the PRINTJOB command in your code.

Don't confuse _peject with the EJECT command, which causes dBASE IV to go to the next page on the printer by sending a form feed or line feeds, depending on the setting of _padvance.

## Options

"BEFORE" — eject sheet before printing the first page.

"AFTER" — eject sheet after printing the last page.

"BOTH" — eject sheet both before the first page and after the last page.

"NONE" — eject sheet neither before the first page nor after the last page.

## See Also

_padvance, EJECT, EJECT PAGE, PRINTJOB/ENDPRINTJOB _pepage specifies the ending page for a printjob.

## Syntax

_pepage = <expN>

## Default

The default page number on which to end printing is 32,767.

## Usage

The _pepage system memory variable specifies that pages with numbers greater than _pepage are not to be output in a printjob.

The allowed range of values for this variable is 1 through 32,767. The value of _pepage cannot be less than that of _pbpage (described earlier in this chapter).

## Tip

The three system variables _pageno, _pbpage, and _pepage should be consistent. For example, if _pbpage = 3, _pepage = 5, and _pageno = 10, dBASE IV won't print any pages, because the current page number (10) is outside the range of pages to be printed (3 to 5).

The value of _pageno should be less than or equal to the value of _pepage. For the example just given, _pageno should be less than or equal to 5 before the printjob will begin. You can use this feature to print just part of a document by suitably setting _pageno, _pbpage, and _pepage. To print a single page, for example, set both _pbpage and _pepage equal to the number of the page you want to print.

## See Also

_pageno, _pbpage, PRINTJOB/ENDPRINTJOB

# _pform

_pform returns the name of the current print form file or activates a print form file, which contains certain print settings.

## Syntax

_pform = "<print form filename>"

## Default

The default for _pform is a null string.

## Usage

A print form (.prf) file is a binary file that contains information regarding print settings. In particular, it contains the settings for the following system memory variables: _padvance, _pageno, _pbpage, _pcopies, _pdriver, _pecode, _peject, _pepage, _plength, _ploffset, _ppitch, _pquality, _pscode, _pspacing, and _pwait. It also contains the destination DOS filename, if one was specified when the print form file was saved.

The report and label generators optionally store these settings to a print form file when the object (report or label) is saved. The settings from this file are loaded automatically whenever you modify or print the object from its design screen or from the Control Center.

When you use the REPORT FORM or LABEL FORM command, however, print form file settings are not automatically loaded. If you want the settings from a particular print form file rather than the current environment's settings to be used when issuing the REPORT FORM or LABEL FORM command, you should assign the name of that print form file to the _pform system memory variable.

## Example

To assign the _pform system memory variable and print a report with the REPORT FORM command:

```
. _pform = "ReportA"
. REPORT FORM Report
```

## See Also

CREATE/MODIFY LABEL, CREATE/MODIFY REPORT, LABEL FORM, REPORT FORM

# _plength

_plength sets the length of the output page.

## Syntax

_plength = <expN>

## Default

The default page length is 66 lines.

## Usage

The total page length is the number of lines from the top of the page to the bottom of the page.

You can assign this variable an integer value ranging from 1 through 32,767.

You don't need to explicitly define the top and bottom margin of a page. By setting _plength, you determine the total page length. Then use ON PAGE to control where headers, footers, and text print on the page.

**NOTE**
*Figure 1-1 in Chapter 1, "Essentials," shows the _plength setting on a page layout.*

## Example

(_padvance contains an example that uses the _plength system memory variable to change the page length.)

## See Also

_padvance, EJECT, EJECT PAGE, ON PAGE

## _plineno

_plineno assigns the line number for the streaming output, or returns the current line number.

### Syntax

_plineno = <expN>

### Default

The default print line number is 0.

### Usage

_plineno works on the data stream that is being output. This *streaming output* is produced by all commands that create output, except the @, @...TO, and EJECT commands. Streaming output destinations may be controlled by the SET CONSOLE, SET PRINTER, and SET ALTERNATE commands, and by the TO PRINTER/TO FILE <filename> options of commands such as LIST/DISPLAY.

dBASE IV keeps track of the current line number in your document. It increments this variable as it formats and prints text.

After changing the paper position in the printer, you can set the _plineno variable to the number where the printhead is positioned. You can assign _plineno an integer value ranging from 0 to (_plength – 1).

Unlike PROW(), _plineno is effective even when you turn off the printer and scroll the output on the screen. Unlike PROW(), _plineno cannot exceed _plength.

Don't confuse the _plineno system memory variable with the LINENO() function, which returns the line number about to be executed in a program.

### See Also

_plength, EJECT PAGE, ON PAGE, PCOL(), PROW()

_ploffset sets the page left offset for printed output only.

## Syntax

_ploffset = <expN>

## Default

The default page left offset is 0.

## Usage

The page left offset is measured from the left edge of the paper. _lmargin begins at the end of _ploffset. dBASE IV prints spaces at the left edge of each line, corresponding to the number defined for this system variable. Use _ploffset to move all text on a page to the right. This feature makes it easy to adjust text on the printed page, if the paper is slightly off center in the printer.

You can assign this variable a value from 0 to 254.

The SET MARGIN command is equivalent to the _ploffset system variable, not to _lmargin, and is automatically adjusted to match _ploffset changes.

The _ploffset and _lmargin settings are independent. For example, suppose you set up a report with a _ploffset of 10 and an _lmargin of 5 and you have indented one section of the report assigning _indent a value of 10. If you subsequently decide to decrease the _ploffset to 5, the relative indentation afforded by _lmargin and _indent is not lost.

**NOTE**
*Figure 1-1 in Chapter 1, "Essentials," shows a typical _ploffset on a page layout.*

## See Also

_lmargin, SET MARGIN

# _ppitch

_ppitch sets the printer pitch or returns a string showing the currently defined pitch.

## Syntax

_ppitch = "pica"/"elite"/"condensed"/"DEFAULT"

## Default

The default _ppitch is "DEFAULT", the pitch defined by your printer's DIP switch settings or setup code.

## Usage

_ppitch sets the pitch (characters per inch) on the printer by sending an escape code appropriate to the active printer driver. (Use _pdriver to select the printer driver.)

## Options

"PICA" — 10 characters per inch.

"ELITE" — 12 characters per inch.

"CONDENSED" — approximately 17.16 characters per inch.

"DEFAULT" — unmodified printer pitch setting.

If you have not set _ppitch in the current work session, the "DEFAULT" pitch is the actual pitch setting of your printer.

Any option that you specify must be supported by the currently selected printer driver.

## Examples

To find out the currently defined pitch, enter:

```
. ? _ppitch
```

The following report program module sets the header and footer to ELITE pitch, prints the column headings in PICA, prints the body of the report in CONDENSED mode, and then resets the pitch to PICA when finished. The procedures Header, Col_head, Body, and Footer are not provided in disk files with your dBASE IV package.

```
_ppitch = "ELITE"

DO WHILE .NOT. EOF()
   DO Header
      _ppitch = "PICA"
   DO Col_head
      _ppitch = "CONDENSED"
   DO Body
      _ppitch = "ELITE"
   DO Footer
ENDDO

_ppitch = "PICA"
```

## See Also

_pdriver, _pquality

# _pquality

_pquality selects quality or draft mode on the printer, or returns a logical condition showing the currently defined print mode.

## Syntax

_pquality = <condition>

## Default

The _pquality default is false (.F.), that is, draft mode.

## Usage

If _pquality is set to true (.T.), dBASE IV selects letter-quality or near letter-quality mode, whichever one the printer supports. If _pquality is set to false (.F.), dBASE IV selects draft mode. (Use _pdriver to select the correct printer driver, so that dBASE IV sends the proper codes to the printer.)

Letter-quality mode produces printed copy of a higher print quality (resolution), while draft mode prints more quickly.

## Tip

By changing the quality setting on the printer itself, you or a user of your application could get the print quality out of sync with the dBASE IV _pquality setting. In this case, the printer setting would take precedence. To make sure the two settings match, issue a _pquality statement from the dot prompt or in your program.

## See Also

_pdriver, _ppitch

# _pscode

_pscode provides the starting control codes for a printjob.

## Syntax

_pscode = <expC>

## Default

The default _pscode is a null string.

## Usage

Sometimes you want to print a particular report in a different typestyle than you customarily use. For example, you might want to print a wide report in condensed mode. _pscode sends the ASCII printer control code which starts the alternate style at the beginning of a printjob. (_pecode, described earlier in this chapter, ends the alternate typestyle.)

See the ??? command and your printer manual for information about printer control codes and control character specifiers.

Set _pscode from the dot prompt or before the PRINTJOB command in your program. dBASE IV sends the codes defined by _pscode to the printer when it encounters the PRINTJOB command.

The ?/?? and ??? commands also send printer control codes to the printer. In general, use ?/?? (STYLE option) to change typestyles on individual lines of text, ??? to change typestyles on a broader basis within a document, and _pscode and _pecode to define the overall typestyle for a printjob.

## Example

This routine sets a Hewlett-Packard LaserJet to landscape orientation at the beginning of a printjob. (See PRINTJOB/ENDPRINTJOB for information about how to code a printjob.)

```
_pscode = "{27}{38}{108}{49}{79}"
```

## See Also

_pecode, ?/?? (STYLE option), ???, PRINTJOB/ENDPRINTJOB

# _pspacing

_pspacing establishes the line spacing for output.

## Syntax

_pspacing = 1/2/3

## Default

The default line spacing is single spacing (_pspacing = 1).

## Usage

The _pspacing system variable determines the line spacing of the streaming output. Line spacing is the number of lines between lines of output. _pspacing accepts an integer value from 1 through 3.

_pspacing also affects the height of a box. If a box is defined with a height of 10, _pspacing = 2 prints the box with a height of 20.

For occasional spacing between lines, use the ? command.

## Options

1 — Single spacing: no blank lines.

2 — Double spacing: one blank line.

3 — Triple spacing: two blank lines.

## Example

To change the output from a LIST command to double-spaced lines, enter the following in a program file:

```
_pspacing = 2
LIST TO PRINT
```

## See Also

?/??, LIST/DISPLAY commands

# _pwait

_pwait determines whether the printer stops after printing each page.

## Syntax

_pwait = <condition>

## Default

The default value for _pwait is false (.F.).

## Usage

The _pwait system variable provides support for cut-sheet printing. When _pwait is true (.T.), dBASE IV stops the printer after each page is ejected, and gives you a chance to change the paper in the printer.

This occurs when the printer ejects a page either because of an EJECT command, or because the number of printed lines has exceeded the maximum defined by _plength.

When using _pwait with printjobs, define the variable in your code before the PRINTJOB command.

## Example

To make printing pause between pages, set _pwait to true (.T.):

```
. _pwait = .T.
```

## See Also

_plength, EJECT, EJECT PAGE

# _rmargin

_rmargin defines the paragraph right margin for the output of the ?/?? command when _wrap is true (.T.).

## Syntax

_rmargin = <expN>

## Default

The default right margin is 80 characters.

## Usage

The right margin is the first column past the text printed on a given line.

The minimum value of _rmargin must be either (_lmargin + 1), or (_lmargin + _indent + 1), whichever value is larger. _indent, of course, may be a negative number. The maximum value of _rmargin is 255.

For example, if the left margin and indentation are both set to 5, the right margin must be greater than 10 (to allow for at least one column of printed output).

_rmargin has no effect unless _wrap is set to true (.T.).

**NOTE**
*Figure 1-1 in Chapter 1, "Essentials," shows a typical _rmargin on a page layout.*

## Example

You can use _rmargin to condense the width of a paragraph at the dot prompt or within a program file. The following programming example prints the same text with two different sets of left and right margins:

```
SET TALK OFF

_wrap = .T.
_rmargin = 72
_lmargin = 0

Mtext = "The quick brown fox jumped over the lazy dog. " +;
"When the fox jumped, the dog started wagging its " +;
"tail and then chased the fox."

? Mtext
?

_rmargin = 45
_lmargin = 15

? Mtext
```

The output for this routine looks like this:

```
The quick brown fox jumped over the lazy dog. When the fox jumped, the
dog started wagging its tail and then chased the fox.

               The quick brown fox jumped
               over the lazy dog. When the
               fox jumped, the dog started
               wagging its tail and then
               chased the fox.
```

## See Also

_indent, _lmargin, _ploffset, _wrap

# _tabs

_tabs sets one or more tab stops for screen, printer, or file output printed with the ?/?? command, and also sets the default tab stops for the word wrap editor.

## Syntax

_tabs = <expC>

## Default

Although the default value of _tabs is an empty string, its default behavior is that of "8, 16, 24, 32...".

## Usage

_tabs defines a list of tab stops. When a tab character, CHR(9), is printed with the ?/?? command, it is expanded to the number of spaces required to reach the next tab stop.

The tab list must consist of a series of numbers in ascending order, separated by commas. These are the column numbers for each tab.

You can set tabs whether _wrap is true (.T.) or false (.F.). If _wrap is true (.T.), the tab stop positions equal to or higher than _rmargin are ignored.

You can also define tab stops in the word wrap editor with _tabs. If you do not set _tabs, the word wrap editor defaults to eight spaces per tab stop.

_tabs is equivalent to the TABS setting of the Config.db file.

## Example

To set the tab stops to 5, 20, 36, and 60, enter the following command:

```
. _tabs = "5, 20, 36, 60"
```

## See Also

_indent

Chapter 6, "Configuring dBASE IV," which discusses the Config.db file.

# _wrap

_wrap sets word wrapping between margins on and off.

## Syntax

_wrap = <condition>

## Default

By default, wrap is set to false (.F.).

## Usage

Setting _wrap to true (.T.) causes the output of the ? and ?? commands to wrap within the margins defined by _lmargin and _rmargin. In other words, text which would otherwise go beyond the right margin automatically displays or prints on the next line, breaking between words or numbers.

The system memory variables _alignment, _indent, _lmargin, and _rmargin are effective only when _wrap is set to true (.T.).

## Example

_wrap determines whether or not the margin, alignment, and indent settings are activated:

```
_indent = 3
_lmargin = 10
_rmargin = 30
Mem = "This is a sentence that contains almost sixty characters."
_wrap = .T.
? Mem
```

This code produces the following output:

```
             This is a
          sentence that
          contains almost
          sixty characters.
```

## See Also

_alignment, _indent, _lmargin, _rmargin

# YEAR()

The YEAR() function returns the numeric value of the year from a date expression. The result is always a four-digit number.

## Syntax

YEAR( < expD > )

## Usage

Use this function to index on a date field in combination with a character field such as a name field.

## Examples

If the system date is 05/15/88:

```
. ? YEAR(DATE())
        1988
```

To store the year from the system date to the memory variable:

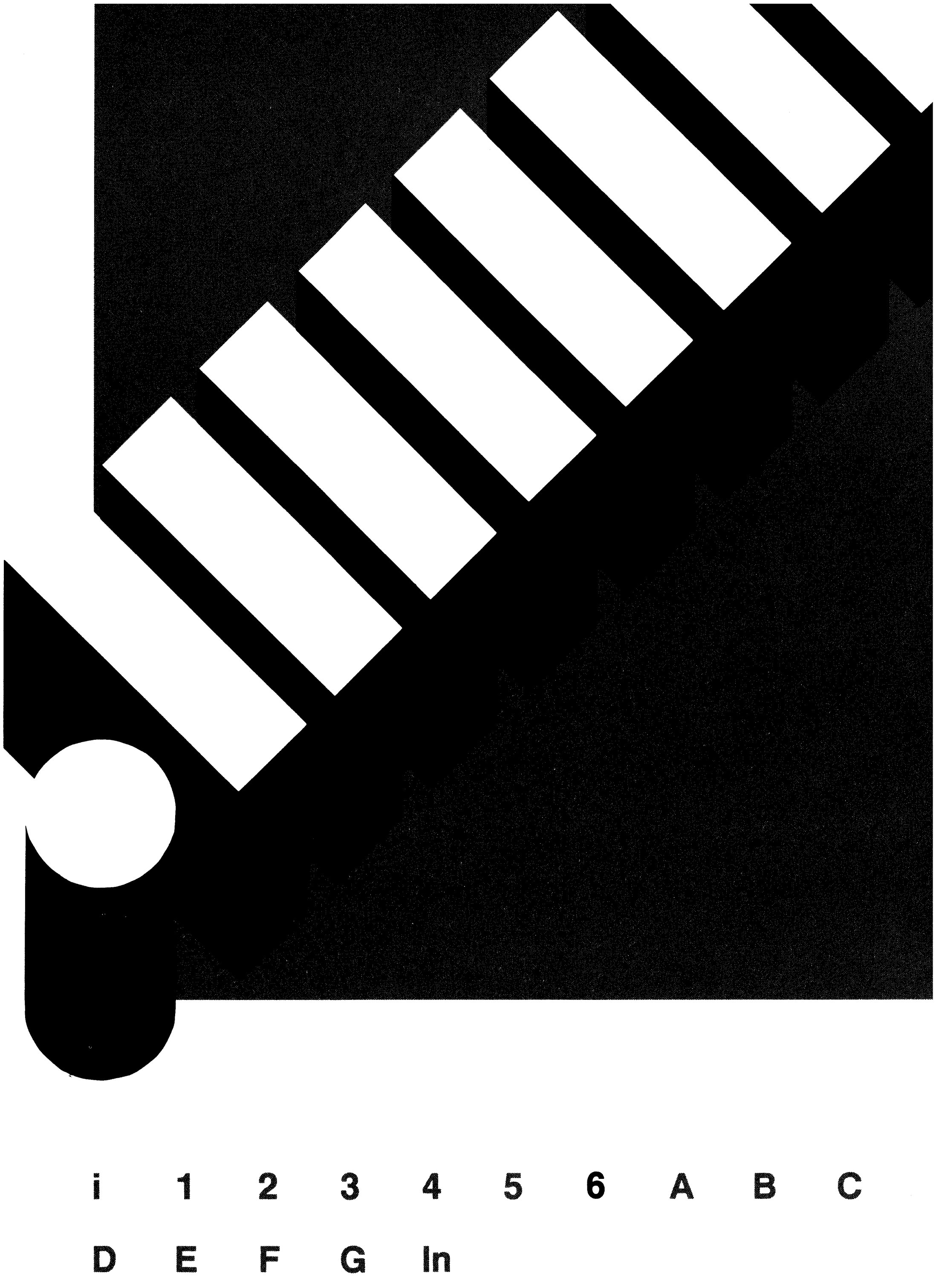
```
. Myear = YEAR(DATE())
        1988

. ? TYPE("Myear")
N
```

To increase the current system year by 50 and store the resulting year to the numeric memory variable Newyear:

```
. Newyear = YEAR(DATE()) + 50
        2038.
```

# Customizing dBASE IV



i 1 2 3 4 5 6 A B C

D E F G In

# Customizing dBASE IV

dBASE IV installs a variety of default configurations. You may alter some of the default settings when you initially install dBASE IV, while others are automatically set for you. Depending on your preferences and your particular hardware, you may want to reconfigure the defaults. This chapter explains how to determine your hardware configuration and change the dBASE IV defaults to meet your needs.

When customizing dBASE IV, you assign values to the operating system (DOS) and to dBASE IV program parameters. You use the Config.sys file to configure DOS and the Config.db file to customize dBASE IV. This chapter covers the changes you can make to Config.sys and Config.db. It also covers the DBSETUP **Tools** menu, which you may use to determine your hardware configuration and disk drive statistics.

For information on installing dBASE IV or customizing DOS, refer to your *Getting Started* booklet. If you need specifics about the **DOS Utilities** menu, see *Using the Menu System*. For detailed descriptions of the SET commands accessible from Config.db, see Chapter 3 of this manual.

## Config.sys

Config.sys is a DOS file that you use to define operating system configuration commands. The BUFFERS and FILES commands, for instance, make certain you are getting the most from dBASE IV's power and data handling capacity.

BUFFERS designates the number of buffers allocated in memory for reading and writing information to and from a disk. FILES refers to the number of files that can be open at one time.

You may already have this file in the root directory of your default drive. If you do not have a Config.sys file or the values in your file are not sufficient when you install dBASE IV, the Install program will create Config.sys or alter the existing file at your request. Otherwise, you can create it with an ASCII text editor. Put each command and its values on a separate line with a ↵ at the end of the line. After you have installed dBASE IV, use MODIFY COMMAND to change Config.sys.

For dBASE IV the Config.sys file should contain:

```
FILES = 40
BUFFERS = 15
```

Enter these values, and save the Config.sys file to the root directory of the disk you boot from.

**NOTE**
*If you are going to use dBASE IV with the 99-file maximum, increase FILES to 99 or greater, keeping in mind the DOS maximum of 255 files. Leave BUFFERS unchanged at 15.*

Whenever you modify or create a Config.sys file, reboot your computer from the DOS prompt with **Ctrl-Alt-Del** so that DOS will use the new values. Refer to your DOS reference manual for more information on the Config.sys file.

## Config.db

Config.db is a special file that loads dBASE IV's program parameters. During the installation of dBASE IV with the DBSETUP or Install program, a Config.db file is copied to the dBASE IV directory. You can change the contents of this file or create new Config.db files with the DBSETUP program, the dBASE IV internal editor, or any ASCII text editor.

Each time you start dBASE IV, it reads the Config.db file and loads the specified program parameters. Therefore, you either must place the Config.db file in the same directory as dBASE IV, or must be sure you can access it through the DOS path from your current directory.

**TIP**
*If you need several different customizations of dBASE IV for different users or applications, create Config.db files in different directories. Edit your Autoexec.bat file to add the dBASE IV directory to the PATH statement. Then you can start dBASE IV from different directories, with different customizations. Refer to your DOS reference manual for more information on the PATH statement and the Autoexec.bat file.*

Use the Config.db file to customize the following parameters:

- Configuration commands
- Memory allocation
- Function key definitions
- Most of the SET commands
- Color settings

## Configuration Commands

dBASE IV uses special configuration commands that you can execute only from the Config.db file; you cannot enter them from the dot prompt. These commands set the operating environment for the screen display on startup, memory allocations, default printer outputs, and the default text editor. Table 6-1 lists these commands with their initial values.

Table 6-1 Configuration commands for Config.db

| Name | Value | Note |
|---|---|---|
| BUCKET | = <expN> | Default is 2K; range, 1K to 31K |
| COMMAND | = <expC> | Any executable dBASE command |
| DO | = <expN> | Default is 20; range, 1 to 256 |
| EEMS | = ON/off | Used to access extended or expanded memory |
| EXPSIZE | = <expN> | Default is 100; range, 100 to 2,000 |
| FASTCRT | = ON/off | Used to eliminate "snow" on the screen |
| FILES | = <expN> | Default is 99; range, 15 to 99 |
| GETS | = <expN> | Default is 128; range, 35 to 1,023 |
| INDEXBYTES | = <expN> | Default is 2K; range, 2K to 128K |
| PDRIVER | = <filename> | Any accessible printer driver file |
| PRINTER | = (see syntax) | Configures 4 printers; 1 to 5 fonts each |
| PROMPT | = <expC> | Up to 19 characters; default is a dot |
| RESETCRT | = ON/off | Used to reset display mode |
| SQLDATABASE | = <SQL database name> | SQL database |
| SQLHOME | = <path name> | Path specification to boot up SQL |
| TEDIT | = <filename> | Any ASCII text editor |
| WP | = <filename> | Any ASCII text editor |

### BUCKET

Use BUCKET to allocate memory blocks in kilobytes, from 1K to 31K. Increase the default value of 2K, if you are creating or using a large format file with multiple screens and PICTURE clauses. To increase the memory allocation to 4K, for example, enter:

```
BUCKET = 4
```

## COMMAND

The installation program, with the following command, alters your Config.db file to automatically start up in the Control Center:

```
COMMAND = ASSIST
```

To display the dot prompt instead, remove this line from the Config.db file.

You can use any valid dBASE IV command with the COMMAND setting. If you set COMMAND = DO <filename>, the named file is executed when you start the program.

When both COMMAND = DO <filename> and SQL = ON are included in your Config.db file, dBASE IV is placed in SQL mode when the program file terminates, unless the program is an SQL program file.

**NOTE**
*You can override the setting of COMMAND by naming a dBASE IV program file from the DOS prompt when you start dBASE. Enter:*

```
dbase <program filename>
```

*dBASE IV first loads the program file and then executes it after a few seconds, or when you press ↵ at the license agreement screen.*

## DO

Use DO to specify the number of nested DO calls in your program. Increase the default value of 20, if you are getting the error message **DOs nested too deep**.

## EEMS

The EEMS command allows dBASE IV to access extended or expanded memory on your computer. This option is not available from DBSETUP, Install, or full-screen SET. If you wish to change this setting, you must modify your Config.db file with a text editor. Some networks require EEMS = OFF. See *Networking with dBASE IV* for more information.

## EXPSIZE

EXPSIZE is a memory buffer that dBASE IV uses to contain expressions that are being compiled. If you are using complex expressions in your commands and are encountering the error message **EVAL work area overflow**, you can increase the default value of 100 bytes.

## FASTCRT

FASTCRT allows you to eliminate "snow" (blurry or white speckled areas) or flickering on the screen. This command should be set to OFF when you are using an IBM PC Color Graphics Adapter (CGA) display adapter, or an adapter patterned after the IBM PC CGA. The CGA adapter normally does not cause snow.

To determine which setting you should use, a test has been provided in DBSETUP. This test is the **Optimize color display** option within the **Modify hardware setup** menu of **Install.**

## FILES

The FILES setting determines the maximum number of files that dBASE IV can have open at one time. This command works in conjunction with the FILES setting of Config.sys. For every file specified in Config.sys, DOS reserves 48 bytes of RAM (39 bytes in DOS 2.1). This value is set when you boot your computer. Note that if you choose to allow DBSETUP to update your Config.sys file during installation, your Config.sys FILES setting becomes 40.

Including FILES in Config.db limits the number of files that you can have open in dBASE IV. dBASE IV is limited to the current Config.sys setting and cannot access more files than the setting in your Config.sys file. For every file specified in Config.db, dBASE IV reserves 20 bytes of RAM.

## GETS

This value sets the number of @...GETs that may be active at one time. The GETS setting is similar to BUCKET; you increase the value for GETS if you are creating or using a large format file with multiple screens. To increase the default from 128 to 300, enter:

```
GETS = 300
```

## INDEXBYTES

Each record in a database file is represented by a node in an index file. Each node contains the key expression value for its associated record and a reference to the next node in the index. As you access an index file, its nodes are loaded into memory as needed. dBASE IV loads the nodes into memory in groups determined by the BLOCKSIZE setting when the index file was created; the default BLOCKSIZE is 512K bytes. The number of index file blocks that can be in memory at one time is determined by the INDEXBYTES value.

If you have large and complex index key expressions, you may benefit by increasing the INDEXBYTES setting. By setting INDEXBYTES to a high value, you can load more nodes into memory at one time and thereby increase your index access time. However, if you have large database files with short index key expressions, you may be wasting time by unnecessarily loading nodes into memory as you traverse the index. Decrease the value of INDEXBYTES if you frequently get the error message **Insufficient memory**.

To increase the default value of 2K to 20K, use the command:

```
INDEXBYTES = 20
```

## PDRIVER

PDRIVER lets you set the driver for your printer. When you install dBASE IV, you specify printers from a list provided by dBSETUP. Printer driver files contain the print codes specific to a printer. You may specify the printer to set as the default printer each time you start dBASE IV. You can change this by entering:

```
    PDRIVER = <filename>
```

in the Config.db file. The <filename> must be a valid printer driver file, which resides in the dBASE IV directory or exists on the current path. Alternately, you can run DBSETUP from the Installation Disk and select a new printer driver file from the list dBASE IV provides. If you do not specify a printer driver with PDRIVER =, dBASE IV defaults to Generic.pr2.

Generic.pr2 uses only the most basic printer control commands. This driver has printing limitations similar to those in previous versions of dBASE. It does not have any printer initialization codes or special print characters. The only print styles available are underline and bold.

**NOTE**
*You can issue a temporary, alternate printer driver command from the dot prompt with the system variable _pdriver. See Chapter 5, "System Memory Variables."*

## PRINTER

You can configure up to four different printer drivers at one time. In addition, you can designate from one to five different fonts for each configured printer. The files containing the drivers and the fonts must be available to dBASE IV when it attempts to access the designated printer. You activate a printer driver from the printer selection pull-down menu when you begin a printjob, or by changing the value of the system memory variable _pdriver.

The syntax for installing or changing printers in the Config.db file is:

```
PRINTER <printer int> = <filename> [NAME <name string>]
    [DEVICE <device string>]
```

The term <printer int> is a number from 1 to 4 and <filename> is the name of the printer driver file. You can use NAME <name string> to describe the printer. This NAME is a character string that will appear on the pull-down print menu from which you select the printer model.

Use the DEVICE option to direct the output to the port your printer uses.

The default printer driver is Generic.pr2, and the default DEVICE port is LPT1.

For example, to install the printer driver for the Hewlett-Packard LaserJet™, use the command:

```
PRINTER 1 = HPLAS100.pr2 NAME "Hewlett-Packard LaserJet"
    DEVICE COM1
```

Install or change the printer fonts for each printer by using this syntax:

```
PRINTER <printer int> FONT <font int> = <begin code>,
    <end code> [NAME <font name string>]
```

The <printer int> term refers to the number of the previously installed printer. The term <font int>, 1 through 5 inclusive, is the number you designate for a specific font. The STYLE option of the ?/?? command corresponds directly with these numbers. You can install up to five different fonts for each installed printer.

The <begin code> and <end code> designations are specific for the font you designated by <font int>. You can obtain the codes from your printer manual. dBASE IV sends these codes to the printer to turn the font on and off. The term <font name string> is the name you select for the font designated by <font int>. This name is displayed on the pull-down menu when you start a printjob.

For example, to install the Roman-8 Symbol Set as the first definable font for the LaserJet, use the command:

```
PRINTER 1 FONT 1 = {Esc}(8U, {Esc}(#@, NAME = "Roman-8 Symbol Set"
```

## PROMPT

The PROMPT setting allows you to change the command line prompt from a dot to up to 19 displayable ASCII characters. Avoid using ASCII decimal values 7, 8, 9, 10, 12, 13, 14, 23, 24, and 27 because these characters typically send commands to your printer.

**NOTE**
*Your printer may use additional characters not listed here. Check your printer manual for possible conflicts when issuing a PROMPT setting.*

If you use the print screen capability often, select a character that your printer can print.

For example, if you include the statement:

```
    PROMPT = Next Command?
```

in your Config.db file, the dot prompt is replaced with:

**Next Command?**

## RESETCRT

Some external programs reset the display mode of your screen when they are run. If RESETCRT is ON, dBASE IV ensures that your screen is reset to the mode it was in prior to running an external program. For example, if your display is set to EGA43 in dBASE IV and you call an external editor that sets the screen to a 25-line display, RESETCRT = ON will reset the display back to 43-line mode when you exit the editor.

### SQLDATABASE and SQLHOME

You can execute dBASE IV and go directly into SQL mode by setting the following command in Config.db:

```
SQL = ON
```

Designate the directory containing your SQL files with:

```
SQLHOME = <path name>
```

If you want dBASE IV to automatically activate an SQL database at startup, designate the database name in Config.db with:

```
SQLDATABASE = <SQL database name>
```

When you install dBASE IV, the DBSETUP program automatically puts SQLHOME and SQLDATABASE in Config.db. This file should typically contain the following commands after installation:

```
SQLHOME = C:\DBASE\SQLHOME
SQLDATABASE = SQLSAMPLES
```

### TEDIT and WP

The internal dBASE IV editor is very powerful. You can easily access it to edit your command or memo fields. You may, however, install other text editors as the default with the TEDIT and WP configuration commands. To use an editor other than the dBASE IV editor, you must have enough extra memory to load the DOS Command.com file and the editor. The editor you specify with TEDIT is used with MODIFY COMMAND; the word processor you specify with WP is used to edit memo fields.

The dBASE IV internal text editor uses less dynamic memory than external editors. It also automatically deletes outdated .dbo files. To duplicate this functionality with an external editor, include the command DEVELOPMENT = ON in your Config.db file.

If the editor you specify is not in the default dBASE IV directory, be sure that its directory is included in your DOS path statement; otherwise, dBASE IV will not be able to find it.

## Memory Block Size Allocation

When you load dBASE IV, you can control the amount of dynamic memory that is allocated to memory variables (memvars), run-time symbols, and compile-time symbols. Use the following commands to do this.

### MVMAXBLKS and MVBLKSIZE

You can set the number of memory variable blocks and size of each block with these commands. The memory variable maximum blocks command (MVMAXBLKS) sets the total number of blocks available. The memory variable blocksize command (MVBLKSIZE) sets the number of memory variables within each block. Every memory variable in a block uses 56 bytes of dynamic memory.

For example, the default block allocation is MVMAXBLKS = 10 and MVBLKSIZE = 50. There can be 50 memvars in each of 10 blocks for a total of 500 memvars. In this case, you have designated up to 32,000 bytes of dynamic memory for memory variable use.

To increase the maximum number of memory variables from 500 to 1,000, add the following line to your Config.db file:

```
MVMAXBLKS = 20
```

This setting gives you a maximum of 56,000 bytes (56 bytes x 1,000 blocks) of dynamic memory available for memory variables.

The actual amount of memory used depends on the number of memvars you declare. When MVBLKSIZE = 100 and you declare your first memvar, dBASE reserves one block containing 100 memvar slots. When you declare memvar number 101, dBASE reserves the second block containing another 100 memvar slots. At this time, a total of 11,200 bytes of memory has been set aside for memvar use.

Array declarations are different. Each array declaration takes one memvar slot. dBASE allocates a special array memory block to hold the values of all the elements in the array. See the DECLARE command in Chapter 2 for more information on array declaration.

### RTMAXBLKS and RTBLKSIZE

These commands are similar in usage to those for memory variable blocks. You can set the maximum blocks allowed for unique names of user-defined memvars and symbols used in a run-time session. Both memvar and run-time memory usage are allocated and deallocated in blocks. You control the maximum number of blocks, and the size of each block, by entering values for RTMAXBLKS and RTBLKSIZE in Config.db.

### CTMAXSYMS

Use this command to set the number of compile-time symbols allocated to compile a program or procedure file. Compile-time symbols are the names of user-defined variables, fields, and procedures. Unlike memvars and run-time blocks, you cannot specify both number and size for compile-time symbol block allocation. You specify the maximum number of symbols needed with the command CTMAXSYMS in Config.db. The symbols used and other information on the compiled program are written to the .dbo file.

Increase the CTMAXSYMS setting if you get the error message **Exceeded maximum number of compile time symbols** when you are attempting to COMPILE a file.

**NOTE**
*MVARSIZE was a dBASE III PLUS Config.db option that is no longer supported in dBASE IV. However, if your Config.db file has an MVARSIZE line, dBASE IV will ignore it.*

Table 6-2 shows the range of values for these settings.

Table 6-2 Memory blocksize allocation for Config.db

| | **Default Value** | **Minimum Value** | **Maximum Value** |
|---|---|---|---|
| MVMAXBLKS = | 10 | 1 | 150 |
| MVBLKSIZE = | 50 | 25 | 100 |
| RTMAXBLKS = | 10 | 1 | 150 |
| RTBLKSIZE = | 50 | 25 | 100 |
| CTMAXSYMS = | 500 | 1 | 5,000 |

Whenever possible, you should use small values for the blocksize to conserve dynamic memory. If you do not have enough system memory to maintain the number of memory variables defined, the error message **Not enough memory to allocate for memory variables** appears.

## Function Keys

You can program the function keys **F2** through **F10**, **Shift-F1** through **Shift-F9**, and **Ctrl-F1** through **Ctrl-F10**. Function keys **F1** and **Shift-F10** are not programmable. Use them to access the Help screens (**F1**) and the macro menu (**Shift-F10**).

The syntax you use to define a function key in Config.db is as follows:

<key label> = <expC>

Table 6-3 shows the default settings for the function keys. You may use either the SET full-screen menu command, the SET FUNCTION TO command, or Config.db settings to program these keys. You can see the current setting of any function key by entering LIST/DISPLAY STATUS at the dot prompt. The default settings for the function keys used with the Control Center are also on the keyboard template.

Table 6-3 Function key defaults

| **Key Label** | **Default** |
|---|---|
| F1 | HELP; * |
| F2 | ASSIST; |
| F3 | LIST; |
| F4 | DIR; |
| F5 | DISPLAY STRUCTURE; |
| F6 | DISPLAY STATUS; |
| F7 | DISPLAY MEMORY; |
| F8 | DISPLAY; |
| F9 | APPEND; |
| F10 | EDIT; |
| SHIFT-F10 | MACRO MENU * |

*Not programmable

## SET Commands

You can include program parameters for most of the SET commands in Config.db. The Config.db file uses an equal sign between the keyword of the SET command and the selected setting. For example, SET STATUS ON becomes STATUS = ON.

The directives contained in the Config.db file are the defaults that you want each time you start dBASE IV. The SET commands that you enter at the dot prompt or through the **Settings** menu are temporary. They remain in effect only until another SET command changes them, or until you quit dBASE IV.

Table 6-4 lists the SET commands that can be implemented from the Config.db file. The default values established when you first install dBASE IV are shown in all capital letters. Refer to Chapter 3 for more information on using the SET commands.

Table 6-4 Set commands accessible from Config.db

| SET Command | Argument | Remarks |
|---|---|---|
| ALTERNATE = | on/OFF | |
| ALTERNATE = | <filename> | When set to a filename, ALTERNATE is automatically turned on. |
| AUTOSAVE = | on/OFF | Saves to disk after each I/O operation. |
| BELL = | ON/off | |
| BELL = | frequency, duration | Frequency: default, 512Hz; range, 19 to 10,000 cycles/second. Duration: default, 2 ticks; range, 2 to 19. |
| BLOCKSIZE = | <expN> | Default is 1 (512K); range, 1 to 32. |
| BORDER = | SINGLE/double /panel/none /<border definition string> | Default is a single line. |
| CARRY = | on/OFF | |
| CATALOG = | on/OFF | |
| CATALOG = | <filename> | |
| CENTURY = | on/OFF | |
| CLOCK = | on/OFF | |
| CLOCK = | <row>, <column> | Screen row and column; default is 0,69. |
| COLOR = | ON/OFF | Default set during installation. |
| COLOR = | (see "Color Settings" below) | |
| CONFIRM = | on/OFF | |
| CONSOLE = | ON/off | |
| CURRENCY = | <expC> | Default is $. |
| CURRENCY = | LEFT/right | |

(continued)

Table 6-4 Set commands accessible from Config.db (*continued*)

| SET Command | Argument | Remarks |
|---|---|---|
| DATE = | AMERICAN/ansi /british/french /german/italian /japan/usa /mdy/dmy/ymd | |
| DEBUG = | on/OFF | |
| DECIMALS = | <expN> | Default is 2; range, 0 through 18. |
| DEFAULT = | <expC> | Sets the default drive name. |
| DELETED = | on/OFF | |
| DELIMITERS = | on/OFF | |
| DELIMITERS = | <expC> | Default is a colon(:). |
| DESIGN = | ON/off | |
| DEVELOPMENT = | ON/off | |
| DEVICE = | SCREEN/printer /file <filename> | |
| DISPLAY = | MONO/COLOR /EGA25/EGA43 /MONO43 | Default set during installation. |
| ECHO = | on/OFF | |
| ENCRYPTION = | on/OFF | Valid only when PROTECT is used. |
| ESCAPE = | ON/off | |
| EXACT = | on/OFF | |
| EXCLUSIVE = | on/OFF | |
| FULLPATH = | on/OFF | |
| FUNCTION = | <expN>, <expC> | Function key label, command. |
| HEADING = | ON/off | |
| HELP = | ON/off | |
| HISTORY = | ON/off | |
| HISTORY = | <expN> | Default is 20; range, 0 to 16,000. |
| HOURS = | 12/24 | Default is 12. |

(*continued*)

Table 6-4 Set commands accessible from Config.db (*continued*)

| SET Command | Argument | Remarks |
|---|---|---|
| INSTRUCT = | ON/off | |
| INTENSITY = | ON/off | |
| LOCK = | ON/off | |
| MARGIN = | <expN> | Default is 0; range, 0 to 254. |
| MEMOWIDTH = | <expN> | Default is 50; minimum 8, maximum 32,000. |
| MENUS = | ON/off | This command is included for dBASE III PLUS compatibility only. |
| NEAR = | on/OFF | |
| ODOMETER = | <expN> | Default is 1; range, 1 to 200. |
| PATH = | <path list> | Up to 60 characters. |
| PAUSE = | on/OFF | |
| POINT = | <expC> | Default is a period; one character allowed, space and numbers not allowed. |
| PRECISION = | <expN> | Default is 16; range, 10 to 20. |
| PRINTER = | on/OFF | |
| PRINTER = | PRN/lpt1/lpt2 /lpt3/com1 /com2 | Set during installation. |
| REFRESH = | <expN> | Default is 0; range, 0 to 3,600. |
| REPROCESS = | <expN> | Default is 0; range, −1 to 32,000. |
| SAFETY = | ON/off | |
| SCOREBOARD = | ON/off | |
| SEPARATOR = | <expC> | Default is a comma (,). |
| SPACE = | ON/off | |
| SQL = | on/OFF | SQL mode is activated after the COMMAND = option terminates, unless an SQL program is executed. |
| STATUS = | on/OFF | The Install program adds this command to your Config.db file and changes the value to ON. |
| STEP = | on/OFF | |

(*continued*)

Table 6-4 Set commands accessible from Config.db (*continued*)

| SET Command | Argument | Remarks |
|---|---|---|
| TABS = | <expC> | Default is an empty string (" "). TABS sets the initial value of the _TABS system memory variable. |
| TALK = | ON/off | |
| TRAP = | on/OFF | |
| TYPEAHEAD = | <expN> | Default is 20; range, 0 to 32,000. |
| UNIQUE = | on/OFF | |
| VIEW = | <query filename> / <view filename> | |

## Color Settings

You have several options to consider when setting the colors you want to use with dBASE IV. The SET COLOR command description in Chapter 3 lists the color combinations available. If your computer has a color and a monochrome monitor, specify which monitor dBASE IV is to use with the command:

```
COLOR = ON/OFF
```

dBASE IV senses your hardware configuration. After determining which monitor type you are currently using, dBASE IV automatically switches COLOR to the appropriate setting.

**NOTE**
*If you include a COLOR setting that does not match the monitor you are using, a blank screen will be produced when dBASE IV is run.*

This setting is automatically written into the Config.db file when you install dBASE IV. Its argument depends on your answer to the prompt for mono or color screen.

You can set text and background colors in Config.db by using the following syntax:

```
COLOR = [<standard>][,[<enhanced>][,[<perimeter>]
        [,[<background>]]]]
```

You may also set the following color options in Config.db:

```
COLOR OF NORMAL =
COLOR OF TITLES=
COLOR OF MESSAGES =
COLOR OF BOX =
COLOR OF INFORMATION =
COLOR OF HIGHLIGHT =
COLOR OF FIELDS =
```

See the SET COLOR command in Chapter 3 for a complete discussion on setting color attributes.

## The DBSETUP Tools Menu

The **Tools** menu of DBSETUP provides information you can use for optimizing your system to improve dBASE IV's performance. It also gives you helpful information on some of the technical parameters of your computer.

### Display Disk Usage

The **Display disk usage** option presents information about the disk layout, allocation scheme, and usage for the current default drive. To change the default drive, you must select **Set default drive:directory** from the **DOS** menu. When you select **Display disk usage**, you get a screen displaying information about:

- **Info for disk**: the disk being analyzed.
- **Sector size in bytes**: the amount of data that can be stored in each sector.
- **Number of surfaces**: the number of surfaces on the disk that can be written to.
- **Sectors per track**: the number of sectors in each track.
- **Sectors/Cluster**: the number of sectors per cluster.
- **Cluster size in bytes**: the smallest amount of space that can be used to store a file.
- **Number of tracks**: the number of tracks on the current disk.
- **OEM name**: the operating system used to format the disk.
- **Root capacity in files**: the maximum number of files that can be stored in the root directory.

- Disk usage in sectors, bytes, and, if appropriate, clusters. The disk usage section of the **Display disk usage** screen displays information about:
  - **DOS system area**: space locked out by the formatting process and used by the operating system.
  - **DOS boot area**: the amount of space used by the on-disk program that starts loading DOS into memory.
  - **File allocation table**: the amount of space used by the file allocation table (FAT), which keeps track of where the files are on the disk.
  - **Root directory**: the amount of space used by the root directory.
  - **Space in use**: the amount and percentage of disk space used by all the files and subdirectories.
  - **Locked out**: the number of sectors and percentage of disk space not usable due to bad sectors.
  - **Available**: the amount and percentage of disk space available.

## Test Disk Performance

The **Test disk performance** option shows you the elapsed time and transfer speed for writing and reading both sequential and random records to and from the default disk. To change the default drive, you must select **Set default drive:directory** from the **DOS** menu.

When you select **Test disk performance**, the top of the screen shows information about the file used for the disk performance tests. DBSETUP creates and deletes this file. The four tasks performed during this test are:

- **Writing sequential records**
- **Reading sequential records**
- **Writing random records**
- **Reading random records**

Each test displays the number of bytes, elapsed time, and the transfer speed to and from the disk as the test progresses. As each test is performed, the updated information displays on the screen. Each test is executed automatically until all tests are complete.

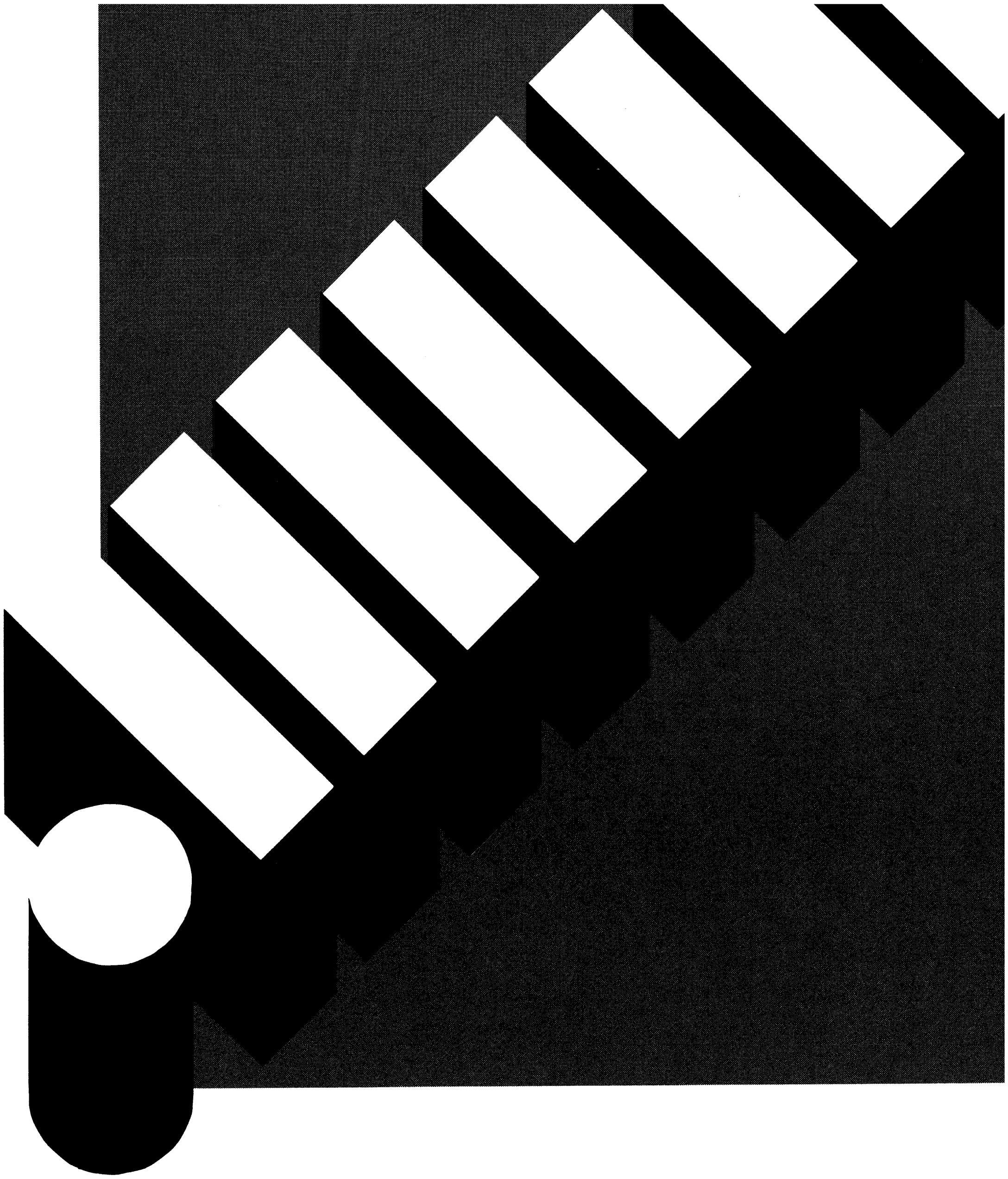
This disk performance test is most informative when used over time. You should run this test periodically to compare figures from one test to another. The fastest times will be when the test is run on an empty directory. Ideally, you should run this test on an empty directory and then, as the directory fills up, run the test again to compare performance figures. As a general rule, the file allocation table has become highly fragmented if the performance drops by 50% from the best time. You should then consider using a utility program that will optimize your disk.

## Review System Configuration

The **Review system configuration** option displays information about your hardware. This display is helpful if you are unfamiliar with the configuration of your computer. This option covers the following items:

- **Chip**: the machine's central processor.
- **ROM BIOS Date**: identifies the ROM version. This option is not available on all machines.
- **Floppy disk drives**: the number of floppy disk drives.
- **Total memory**: total amount of memory available for use by DOS.
- **RS-232 serial ports**: the number of serial ports.
- **Parallel ports**: the number of parallel ports.
- **Math co-processor**: indicates if a math co-processor, used to speed up mathematical operations, is present. dBASE IV takes advantage of a co-processor for all mathematical calculations.
- **Direct Memory Access present**: used to transfer data to and from the computer's memory without passing through the CPU. Direct Memory Access can speed up your computer's overall performance, because it allows the disk drive to read or write without involving the CPU.
- **Game adapter present**: indicates if a game adapter is present.
- **Initial video mode**: video mode set from the internal switch settings.
- **Current video mode**: dBASE IV and DBSETUP read the current display mode via internal hardware interrupt 16 (INT 10H).

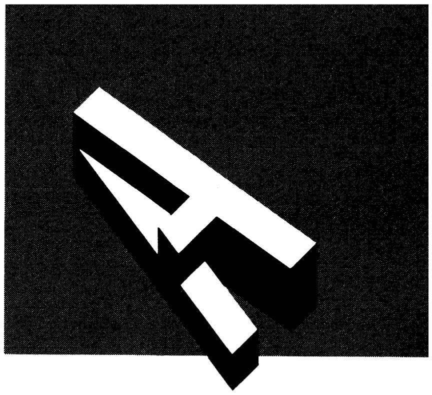
# Appendixes



i 1 2 3 4 5 6 A B C

D E F G In

# Error Messages



This appendix contains all dBASE IV error messages in alphabetical order. The error number associated with each message is shown in square brackets at the end of the message. These numbers are not displayed on the screen with the error message; however, you can trap them with the ON ERROR command and use the ERROR() function to return the error number.

Messages without error numbers originate from the Control Center and cannot be trapped.

Corrective actions are suggested whenever possible.

**A Replace query must have at least one WITH operator** **[370]**
You tried to run an update query (replace) and there is no WITH statement.

**A UNIQUE aggregate must be the only aggregate in a QUERY** **[324]**
An aggregate QUERY can have only one UNIQUE in it.

**ACOS(): Out of range** **[293]**
The value of the expression you used with the arccosine function is out of the range of $-1.0$ to $+1.0$.

**ALIAS expression not in range** **[232]**
You are using an alias expression out of the range of 1 to 10.

**ALIAS name already in use** **[24]**
You attempted to USE a database file that is already open, has the same alias, or has a name or alias within the default range of A through J.

**ALIAS not found** **[13]**
You attempted to SELECT a database area outside the range of A through J or 1 to 10, or used an undefined alias.

**All allowed slots have been filled**
You have used the maximum number of indexes allowed: 10 .ndx and 47 .mdx.

**All database files must be closed before using PROTECT** **[173]**
Close all databases in use before attempting to use PROTECT.

**ASIN(): Out of range** **[291]**
The value of the expression you used with the arcsine function is out of the range of $-1.0$ to $+1.0$.

**ATAN(): Out of range** **[294]**
The allowed range of angles in radians for the arctangent function is between $+\pi/2$ and $-\pi/2$.

**Bad array dimension(s)** **[230]**
This message indicates that you either used an illegal value (such as a zero) during the declaration of an array, or you declared an array that exceeds 1,023 elements.

**Bad EXPRESSION** **[40]**
The syntax of an expression is illegal in a DO WHILE, IF, or CASE command.

**Bad PROCEDURE name** **[32]**
You are trying to use a procedure that dBASE IV cannot find. Check the full DOS file specification of this procedure file.

**BAR position must be a positive number** **[167]**
You must define bars of a menu or popup using positive numbers only.

**Beginning of file encountered** **[38]**
You are attempting to do a backward search in a database file and you are already at the beginning of the database.

**Branching must end before the first @ command** **[308]**
In a format file, the first initialization section can contain any command; but any control structure such as an IF must be terminated by an ENDIF before the second section, which contains only @ commands, starts. Check your format file and terminate program branches before the beginning of the @ commands section.

**Calculated field requires an expression**
You tried to create a calculated field without an expression. You pressed **Ctrl-End** in the validation submenu, and there is no expression entered.

**Calculated fields in the VIEW must not be empty** **[323]**
You deleted the expression of a calculated field that is in the view.

**Cannot add an empty or erroneous calculated field to the view**
You tried to add an invalid calculated field to the view.

**Cannot add a group band to this type of band**
This occurs when trying to add a group band to a band that is not a page header band, report intro band, or group intro band.

**Cannot add fields to the view in an update query**
You tried to add fields to the view when there is an update command in the pothandle of a file skeleton.

**Cannot append in column order** **[147]**
You tried to APPEND from a MultiPlan (SYLK format) spreadsheet with rows not in ascending order. Check to see that the spreadsheet columns are in order before reissuing this command.

**Cannot clear menu in use** **[176]**
A menu active on screen cannot be cleared with the CLEAR MENU or RELEASE MENU command.

**Cannot clear popup in use** **[177]**
A pop-up menu active on the screen cannot be cleared with the CLEAR POPUP or RELEASE POPUP command.

**Cannot close database when transaction is in process** **[185]**
You must complete a transaction and issue an END TRANSACTION command, or perform a ROLLBACK before you can close a database involved in a transaction.

**Cannot close index files when transaction is in process** **[187]**
You must complete a transaction and issue an END TRANSACTION command, or perform a ROLLBACK before you can close index files involved in a transaction.

**Cannot create a link in the pothandle**
The cursor was in the pothandle when you selected **Create link** by pointing (query design).

**Cannot create more than 20 calculated fields**
You tried to add the 21st calculated field to the calculated field skeleton.

**Cannot create! SQL table exists with same name** **[374]**
You are trying to create a .dbf file, and an SQL table exists with the same name. Select a different database filename.

**Cannot delete SQL created TAG: < tagname >** **[376]**
You may delete an SQL created TAG only during interactive SQL mode.

**Cannot enter SUM or AVERAGE in a character field**
You cannot use SUM or AVERAGE in a character field in QBE.

**Cannot erase open file** **[89]**
Close database or index files before attempting to ERASE them.

**Cannot erase a read-only file** **[336]**
You may not erase a file to which you have read-only access. See your system administrator about your file privilege level.

**Cannot execute this command while transaction is in process** **[186]**
See the description of the BEGIN TRANSACTION command for a list of commands that are not allowed during a transaction.

**Cannot go to Browse/Edit if errors exist in the query design** **[344]**
This message occurs when you attempt to go to Browse/Edit when there is an error in the query design.

**Cannot have more than one aggregate followed by UNIQUE**
You have more than one aggregate followed by a UNIQUE clause.

**Cannot have more than one aggregate operator in a column** **[380]**
You have more than one aggregate operator in a column.

**Cannot have more than one GROUP BY in a column** **[339]**
You entered GROUP BY twice in one column.

**Cannot insert. Expression length at limit**
You cannot insert an expression into a database field if the new expression exceeds the field length.

**Cannot JOIN a file with itself** **[139]**
You can only join two different database files. You need to name another database in USE.

**Cannot link a file skeleton to itself** **[345]**
You tried to link two fields in the same file skeleton.

**Cannot load more than eight files in query design**
You can use a maximum of eight file skeletons in QBE.

**Cannot modify SQL table** **[375]**
You cannot modify an SQL table in dBASE IV mode; you must be in interactive SQL mode.

**Cannot move empty field**
This is a Control Center database design error. You get this message if you press **F7** on a blank field in the sort table.

**Cannot MOVE or COPY without a defined selection**
You tried to move or copy without first defining an extended selection.

**Cannot move the cursor beyond bottom of window**
You tried to move the cursor off the pop-up window for the first time. If you try to move the cursor in the same direction again, the window is closed.

**Cannot move the cursor beyond left edge of window**
You tried to move the cursor off the pop-up window for the first time. If you try to move the cursor in the same direction again, the window is closed.

**Cannot move the cursor beyond right edge of window**
You tried to move the cursor off a pop-up window for the first time. If you try to move the cursor in the same direction again, the window is closed.

**Cannot move the cursor beyond top of window**
You tried to move the cursor off the pop-up window for the first time. If you try to move the cursor in the same direction again, the window is closed.

**Cannot nest transactions** **[198]**
Transactions cannot be nested inside other transactions. Issue an END TRANSACTION command before starting a new transaction.

**Cannot open <filename> resource file** **[398]**
This specified resource (.res) file cannot be read. This is a fatal error and results in the termination of dBASE IV. You may try to locate this .res file on your original program diskettes and copy it from the diskette to the directory from which you start dBASE IV on your hard disk. If you cannot locate it, call the Ashton-Tate Software Support Center for assistance, as dBASE IV will not run without this file.

**Cannot overwrite SQL TAG: <tagname>** **[377]**
You may not overwrite an SQL created TAG except in interactive SQL mode.

**Cannot place a link in a field that has an error** **[346]**
You are trying to link to a field that has an error.

**Cannot recover the damaged index file** **[364]**
The contents of an index file are damaged beyond repair. Execute a new INDEX command to create a new index for the database file.

**Cannot re-define menu in use** **[174]**
You are attempting to issue a DEFINE MENU command while there is an active menu in memory. Issue a DEACTIVATE MENU command first, then use DEFINE MENU.

**Cannot re-define popup in use** **[175]**
You are attempting to issue a DEFINE POPUP command while there is an active popup in memory. Issue a DEACTIVATE POPUP command first, then use DEFINE POPUP.

**Cannot release memvar in use: < memvar name >** **[395]**
You cannot RELEASE or CLEAR a memory variable that is being used in an @...GET...READ or as a parameter to a PROCEDURE or FUNCTION.

**Cannot select requested database** **[17]**
You have selected an invalid work area number. Use a number from 1 to 10 or a letter from A to J.

**Cannot use a sort priority number higher than nine**
This is a Control Center error. You get this message when you enter a sort priority number that is higher than nine, because you cannot sort on more than nine fields.

**Cannot write to a read-only file** **[111]**
You are attempting to write to a read-only file.

**Cannot write to database due to incomplete transaction** **[201]**
An incomplete transaction has flagged the database as in use. If you did not start a transaction, check to see if the database is involved in a transaction with the COMPLETED() function. If false (.F.), then wait until that transaction is complete. If you had started the transaction, then issue either an END TRANSACTION to save the changes or a ROLLBACK command to undo the changes and clear the integrity flag from the database file.

**Cannot write to transaction log file** **[188]**
The required transaction log file is not open or available. Issue an END TRANSACTION command to abandon the transaction you attempted.

**Catalog has not been established** **[122]**
You must CREATE a catalog before you can use it.

**CHANGE(), not enough memory** **[161]**
This error is returned if there is not enough memory available to perform the CHANGE() function.

**CHANGE(), record locked by another** **[160]**
This error is returned if you attempt to use the CHANGE() function to determine if the record has been changed and another user has the record locked.

**CHR(): Out of range** **[57]**
The argument that you supplied to the CHR() function is either less than 0 or greater than 255. This range represents the IBM extended ASCII character set.

**Column is full (255 character maximum)**
This message is displayed when you select **Create link by pointing** or **Sort on this field** for a column that does not have enough characters to accommodate the length of the example variable or sort directive.

**Column number must be between 0 and either right margin or 255** **[223]**
You have specified a printer column number that is off the page. Reissue the command using a column number within the page width limits.

**Command not allowed in programs** [306]
You have used a command in a program file that is not allowed in programs such (as RESUME).

**Command not allowed in SQL** [307]
This dBASE command is not allowed in SQL mode.

**Command not allowed in a user defined function** [384]
See Chapter 1, "Essentials," for a list of commands allowed in user-defined functions.

**Command not functional in dBASE IV** [93]
You are using a command such as SET MENU or SET DOHISTORY that is not functional in dBASE IV, but has been retained for backward compatibility.

**Command not supported by RunTime** [290]
Certain commands like MODIFY STRUCTURE and CREATE SCREEN are not allowed in a runtime file. You have a command file that contains one of these commands.

**Command not valid in RunTime environment** [348]
The command you are trying to use cannot be used during runtime.

**Command only valid in programs** [260]
You are trying to use a command that is allowed in programs only from the dot prompt. Write a program or a procedure if you want to use this command.

**Command too long, press CTRL-HOME to edit in zoom window** [402]
You can only edit a 254-character command on the command line. If you need a longer command line, press **Ctrl-Home** to edit the command in the zoom window.

**Command will never be reached** [369]
You have a command in your program that cannot be executed because it is located after a RETURN or an EXIT command. Commands located after the end of a program can never be reached or executed.

**Compilation error** [360]
A program file would not compile because of syntax errors. Correct the errors in the program file and recompile.

**Condition box cannot be extended any further**
You have reached the bottom region of the condition box.

**Condition box has not been created yet**
You have selected **Show condition box** from the **Condition** menu when you did not first create the condition box.

**Conflicting data types**
In query design, you have entered an expression that has conflicting data types.

**CONTINUE without LOCATE** [42]
You cannot continue without executing a LOCATE command.

**Control Code exceeds 255** [401]
You cannot have a printer control code that exceeds 255 characters.

**Coordinates are off the screen** [30]
You are using row and column numbers that are off the screen when defining or moving a window. Limits of the screen are 1,1 to 22,79.

**could not be opened** [72]
You have an illegal ALTERNATE filename specified in your Config.db file. Edit the Config.db file and correct the ALTERNATE filename assignment.

**Current printer driver does not support quality** [331]
The selected printer driver does not support letter-quality mode printing or special fonts. Select the correct printer driver from the dot prompt by using the system memory variable _pdriver = filename.pr2, where filename is the correct printer driver.

**Cyclic relation** [44]
You created an endless loop by setting a relation that relates back to the currently selected database.

**Data Error**
There is a bad sector on your hard disk or floppy disk that dBASE IV has attempted to read. This is a critical hardware error that may cause loss of data.

**Data type mismatch** [9]
You are attempting to mix different data types in one operation. Operations that can generate this message are: an expression that contains different data types, a REPLACE that uses a different data type from the one the field contains, a SEEK that does not match the index key data type, or a SORT that tries to use a logical or memo field.

**Database encrypted** [131]
You are attempting to use an encrypted database without going through PROTECT and the log-in screen to enter your password.

**Database not indexed** [26]
You must create an index file before you can use this command.

**Date and time cannot be sized**
You attempted to size a date or time field using **Shift-F7**.

**DBF file in VIEW is not in current directory**
You tried to create a view for a .dbf file that is not located within the current directory.

**DBT file cannot be opened** [41]
The memo file cannot be opened because it is locked or not available.

**Debugger already in use** [347]
You attempted to call the debugger while it is already in use.

**Disk full when writing file: <filename>** [56]
The disk (hard or floppy) you are trying to write to is full. Make room on the hard disk by removing files, or use a new correctly formatted floppy disk.

**Disk I/O error**
Error in reading/writing to disk could be caused by a full disk, hardware problems, or incompatible media.

**Disk is write protected**
You are attempting to write to a write-protected floppy disk. Remove the tab.

**Display mode not available** [216]
The display mode you selected is not available. Bring up the full-screen SET command menu to see the supported selections.

**Display width exceeds 255** [400]
The report display width cannot exceed 255 characters.

**Drive not ready on <drive>**
The drive letter is appended to this message. The drive you are attempting to read from or write to is not ready. Close the floppy disk drive door.

**Duplicate field name**
You entered a duplicate field name when creating/modifying a database structure.

**Duplicate production MDX file** [211]
You have attempted to open a second production .mdx file.

**Duplicate sort number** [322]
During a semantic check, dBASE IV has found an identical sort directive in two different columns (query design).

**Editing condition not satisfied** [171]
The SATISFY expression in an @...SAY, @...GET command was not met.

**Empty structure will not be saved**
You pressed **Ctrl-W** or **Ctrl-End** to save an empty database structure. The **Save and continue** and the **Save and exit** menu options are dimmed if the structure is empty.

**End of file encountered** [4]
You are attempting to do a forward search in a database file and are already at the end of the database, or SKIPped past the end of the file.

**End of file or error on keyboard input** [51]
This is a diagnostic message intended for internal test. If a tester is using a file for keyboard input and encounters a premature EOF, dBASE IV generates this message.

**Environment not correct for rollback** [193]
You are attempting to do a ROLLBACK when there is no BEGIN TRANSACTION command and a transaction log file does not exist.

**Error during processing of ON ERROR command** [394]
There was an error in the command or program executed during exception (ON ERROR) processing; that is, an error occurred during the execution of the <command> in ON ERROR <command>.

**Error in configuration value** [145]
You have an invalid value in the Config.db file. See Chapter 6 for a discussion of controlling configuration parameters from the Config.db file.

**Error in query definition - cannot execute**
You pressed **F2** from the Control Center, on a query file that has errors in it.

**Error in reading log file** [192]
The transaction log file is corrupted or otherwise cannot be read. A successful ROLLBACK is not possible.

**Error on line <number>** **[96]**
This is a compiler message that supplies the line number where the syntax error has occurred.

**Errors in query definition - cannot be executed** **[352]**
You have errors in your query definition; check your syntax and retry your query.

**EVERY allowed with only one of example variable pair** **[318]**
This is a QBE error message. It indicates that a full outer join is not allowed.

**Exceeded maximum compiler nesting level** **[248]**
dBASE IV allows maximum of 64K of compiled code per procedure, and 32K of compiled code in a branch loop. Reduce program loops/procedure sizes.

**Exceeded maximum number of compile time symbols** **[274]**
Compile time symbols refers to the total unique names of memory variables and database fields in any compiled procedure. The default number allotted to CT symbols is 500. You have exceeded the allotted number; use fewer symbols.

Alternately, you may increase this number in the Config.db file by changing the value assigned to CTMAXSYMS. See Chapter 6, "Customizing dBASE IV."

**Exceeded maximum number of compile time symbol references** **[273]**
Compile time symbols refers to the total unique names of memory variables and database fields in any compiled procedure. The default number allotted to CT symbols is 500. You have exceeded the allotted number; use fewer symbols.

Alternately, you may increase this number in the Config.db file by changing the value assigned to CTMAXSYMS. See Chapter 6, "Customizing dBASE IV."

**Exceeded maximum number of procedures** **[397]**
The maximum number of procedures for a single .dbo file was exceeded.

**Exceeded maximum number of runtime symbols** **[272]**
Runtime symbols refer to the total number of memory variables and database fields used in a dBASE session. The default is 500. You have exceeded this limit during a dBASE session. Use fewer symbols.

Alternately, you may increase this number in the Config.db file by changing the values assigned to RTMAXBLKS and RTBLKSIZE. See Chapter 6, "Customizing dBASE IV."

**Extra characters ignored at end of command** **[151]**
There are extra characters at the end of a statement when dBASE IV expects none. These are ignored by the program.

**<field> data type does not match <field>**
This is displayed as a result of a consistency check between the fields in the form, report, or label and the current database file.

**<field> not found in <dbf>**
This is displayed as a result of a consistency check between the fields in the form, report, or label and the current database file.

**Field must be a memo field** **[350]**
You have entered a field name with the APPEND/COPY MEMO <field name> command that is not a memo field type.

**Field name already exists**
This is a Control Center layout editor error. You get this message when a calculated field name or summary field name conflicts with the name of a field that already exists.

**Field name already exists in the view**
You tried to rename a field name with a name that already exists in the view.

**Field name already in use**
You are attempting to use an existing field name for a new field. Pick another name.

**Field name not found**
In the field sort list, you entered a field name that does not exist in the current database file.

**Field name required**
You tried to move out of an empty field when there is a field following the empty field. You are being asked to name the empty field. Leaving an empty field with no field following it prompts the user to save the file or continue.

**Field name too long**
The renamed field in the view skeleton has too many characters. The field is truncated to ten characters.

**Field not found** **[48]**
You are searching for a field that cannot be found in the file in use.

**Field requires a name**
You tried to hide a calculated or summary field that doesn't have a name. Hidden fields must have names.

**Field type must be A, B, C, D, L or M**
You pressed the wrong key while changing the field type.

**Field type must be Y or N**
This is a message that is used in database design. It displays when you press an invalid key in the Index column of the database design structure.

**Fields list too complicated** **[141]**
Reduce the number of fields in the fields list.

**File already exists** **[7]**
You are trying to create or rename a file to an existing filename. Pick another name.

**File already in catalog** **[286]**
This message occurs in the Control Center if you attempt to enter a file into the catalog and that filename is already in that catalog.

**File already open** **[3]**
The file you are attempting to open is already open.

**File cannot be modified because DESIGN mode is set to OFF** [314]
This message usually occurs from application programs that prevent a user from accessing the design modality of dBASE IV. Certain application programs do not let users access design commands such as EDIT, APPEND, BROWSE or to use the **Shift-F2** key combination from the Control Center. If you are in an application program that uses the Control Center as its interface, you should not need to enter into the design mode.

**File catalog empty**
There are no files in the catalog you are attempting to use. Use another catalog or create a catalog and include the files you want in it.

**File does not exist** [1]
The specified file cannot be found. Check the directory for the filename.

**File has been deleted** [49]
You are attempting to open or delete a file that has been deleted.

**File in use by <username>** [372]
In multi-user operation, the file you want to use is locked by the user identified in the error message.

**File in use by another** [108]
The file you want to open is in USE by another user on a multi-user system. Retry later.

**File must be opened in exclusive mode** [110]
You are trying to use a command such as PACK on a file that you did not open for exclusive use. Close and reopen the file with exclusive ON before attempting the command again.

**File not accessible** [29]
You are trying to use a file that is read-only or has some other access restriction placed on it by the operating system (DOS).

**File not an MDX index file** [206]
Create a multiple index file for the database.

**File not in transaction log** [194]
You cannot perform a ROLLBACK on a file that was not recorded in a transaction log file. You must restore this file to its pre-transaction state from backup copies.

**File not linked** [326]
During a semantic check, dBASE IV has found that the files in the query definition are not linked (query design).

**File skeleton cannot be extended any further**
You have reached the bottom of the file skeleton region.

**File was not LOADed** [91]
You must load a binary program file before you can CALL it in a larger program.

**Find not successful** [14]
dBASE IV was unable to find the character string you searched for.

**FIRST allowed with only one example variable pair**
This error occurs in QBE, when you use the QBE keyword FIRST illegally with more than one example pair.

**Format files must be in row major order to be exported**
You must have the screen coordinates for the @ commands in ascending order before you can export this file to PFS:FILE.

**FROM database must be in one of the unselected work areas** **[189]**
You can append or copy from a database only when that database is not in the selected work area.

**Function prohibited in SQL mode: < function name >** **[340]**
The function named in the error message cannot be used in SQL mode.

**General failure**
This message is issued by the operating system to indicate hardware errors in reading/writing to a disk. Check media compatibility; perhaps the disk format is not compatible. Check the filespec for the destination file to make sure that the name of the drive you are writing to is included.

**GROUP BY used without an aggregate operator** **[327]**
During a semantic check, dBASE IV has found the operator "GROUP BY" but not a corresponding aggregate operator (query design).

**Group name has not been defined** **[155]**
You must define a group name before you can use the PROTECT command to encrypt a file.

**Group name has not been established** **[158]**
You must establish a group name before you can use the PROTECT command to encrypt a file.

**Hidden field requires a name**
This is a Control Center layout editor error. You see this message when you try to hide a calculated or summary field that has not been given a name. Give a name to all fields you wish to hide.

**Illegal call to the compiler** **[379]**
You may not call the compiler from a user-defined function, or an ON ERROR condition.

**Illegal character data length**
This is a Control Center design error message. You specified a structure-extended file in CREATE FROM that has a field length not within the 1 through 254 range.

**Illegal data length**
You specified a character-type field length in CREATE or MODIFY STRUCTURE that is not within the 1 through 254 range.

**Illegal decimal length**
This is a Control Center database design error message. The maximum number of decimals allowed is 0 through 18, or a total of 20 characters including the decimal point.

**Illegal field name**
The field name contains too many characters or invalid characters or spaces. Use underscores instead of spaces. Limit names to eight characters or less.

**Illegal field width**
You have entered an illegal field length for the current field in database design.

**Illegal Key Expression** **[386]**
Check your syntax, since the key expression you used is not a legal dBASE IV expression.

**Illegal macro usage** **[266]**
You are using a memory variable with the program macro (the & function) that is not a character type. For example, if you assign a numeric value to a memory variable, such as x = 3, then use x with the & function. X is really a numeric value, and this is an illegal use of a program macro.

**Illegal numeric data length**
A numeric data field can be between 1 and 20 characters. You are trying to exceed this limit.

**Illegal operation. Database structure is empty or has been changed** **[362]**
This is a Control Center database design error message. This message is used in MODIFY STRUCTURE. It displays if you attempt to execute **F2** (EDIT/BROWSE) or **Shift-F2** after you have changed the structure of a database file.

**Illegal value** **[46]**
Using the @ command to draw a box, you specified a TO coordinate that is less than the row 1 column 1 coordinate. This message also occurs with SET TO commands if you specify an invalid number.

**Index damaged. Do REINDEX before using data** **[114]**
The index file associated with the database in USE is damaged. Create a new index with the REINDEX command.

**Index expression is too large (220-char maximum)** **[112]**
The key expression you are INDEXing on exceeds 220 characters. Reduce the field length for one or more of the fields. You may use LTRIM() or RTRIM() to remove blanks.

**INDEX failed - key is not DISTINCT** **[236]**
Select another key expression to INDEX ON.

**Index field must be Y or N**
You pressed the wrong key when selecting the index type in the database file structure.

**Index file does not match database** **[19]**
The index file you are trying to USE does not have a keyword related to the database in USE. If the database in use has no index file, create a new index file.

**INDEX interrupted. Index will be deleted if not completed** **[113]**
You have pressed the **Esc** key during INDEX or REINDEX execution. You may continue or abandon the indexing process by responding to the prompt of this error message.

**Index is too large (100-char maximum)** **[23]**
You are trying to use an expression that exceeds the maximum for an index key. Use a shorter key word.

**Index TAG already exists** **[205]**
The index tag you are trying to use already exists; select another tag.

**Insufficient memory** **[43]**
You attempted a memory allocation operation that failed because you have reached the limits of available RAM.

Add more memory, or remove memory resident utilities and restart dBASE.

**Insufficient space on row**
You are trying to insert text on a row that is full.

**Insufficient space on row, field truncated**
You are trying to insert a field on a row that does not have enough space. The field has been truncated to make it fit in available space.

**Insufficient space on row, position field with cursor keys**
You are trying to add a field to a line that is completely full; there is not enough room to truncate the field to one character.

**Internal editor error. Edit buffer may be damaged**
This is a fatal error that should never occur. If you get this message, please call the Ashton-Tate Software Support Center.

**Internal error: CMDSET():** **[66]**
This error should not occur. This internal error message displays if a programmer calls the internal SET routine with a faulty parameter. The .dbo file is corrupted and contains token code that can be interpreted as an illegal SET command token code.

**Internal error: EVAL work area overflow** **[67]**
This is determined by an internal evaluating routine of dBASE called EVAL. The string you entered on the dot prompt will cause an internal stack overflow. Check the number of nested function calls in the string.

**Internal error: Illegal opcode** **[68]**
This is an internal error. The compiler has generated a number for a command that does not exist.

**Internal error: Internal table overflow** **[69]**
This error indicates that dBASE has run out of room in its parse table while attempting to parse an extremely complicated expression. The solution is to simplify the expression.

Large expressions/fields may require a larger parse table. Use the Config.db parameter EXPSIZE = <expN>. The minimum and default is 100. Try 500.

**Internal error: Unknown command code:** **[65]**
This message can occur if the .dbo file you are using is invalid. You see the decimal representation of the illegal opcode after the colon.

**Internal SQL code generation error** **[311]**
This message can occur if there is an internal inconsistency during SQL code generation which causes a problem with dBASE IV. You should never get this message; if you do, please call the Ashton-Tate Software Support Center.

**Internal: Virtual Stack Overflow** **[199]**
This error occurs if the internal evaluation stack overflows. It can occur if an expression is too complex.

**Invalid box dimensions** **[227]**
When drawing boxes with the @ command, row 2 column 2 must be larger than row 1 column 1.

**Invalid date** **[81]**
You are trying to enter an invalid date into a date field. Press the **Spacebar** to clear this message.

**Invalid DIF character** **[118]**
You are trying to APPEND from a DIF format file that contains an invalid character or a control character.

**Invalid DIF file header** **[115]**
The DIF file that you are trying to import from does not have the correct file header.

**Invalid DIF type indicator** **[117]**
The DIF file that you are trying to import from contains an invalid data type indicator.

**Invalid DIF vector - DBF field mismatch** **[116]**
The DIF file that you are trying to import from has an internal conflict between its header and its data.

**Invalid DOS SET option** **[99]**
You have an invalid DOS SET option in your Autoexec.bat file. Check your MS-DOS or PC-DOS Software Reference Manual.

**Invalid file extension: <.ext>** **[333]**
The file extension you specified is not correct for the file type you are trying to use. Check the extension for the file type, or do not specify an extension and let dBASE IV use its default.

**Invalid function argument** **[11]**
Check the function syntax in Chapter 4, "Functions," for allowed arguments.

**Invalid function name** **[87]**
See Chapter 4, "Functions," for function names.

**Invalid group name for file. Please re-enter** **[159]**
The group name you are trying to use with the PROTECT program is invalid. Use a valid group name.

**Invalid index number** **[106]**
Index files have a range between 0 and 7. You are trying to SET ORDER TO to a number that exceeds this range, or the position is empty.

**Invalid key label** **[317]**
You are trying to assign a function to a key that is not programmable. Check the full-screen **SET** menu to see the available list of programmable function keys.

**Invalid Lotus 1-2-3 version 2.0 spreadsheet** **[297]**
You are trying to import a Lotus spreadsheet that is not version 2. Please check your spreadsheet version. Convert it to a version 2 file using the Lotus conversion utility before trying to import it.

**Invalid macro file header** [356]
This error occurs if the macro file you are reading is corrupted or is not a dBASE IV macro file. It means an inconsistency was detected during the reading of the header of a macro file.

**Invalid operator** [107]
The operator you are trying to use is invalid for the data types it is connecting.

**Invalid password. Please re-enter** [153]
You did not enter your password correctly. Repeat your entry. If this problem persists, see your system administrator.

**Invalid path or filename: <filename> / <path>** [202]
The file specification you are using is incorrect.

**Invalid printer port** [123]
The SET PRINTER TO command is specifying a port that does not exist. Check your configuration to see what ports you have. If you have not redirected your output, the DOS default is LPT1.

**Invalid printer redirection** [124]
You have redirected the print output to a printer other than LPT1, and the new port does not exist.

**Invalid SET expression** [231]
Check the syntax for the supported SET commands.

**Invalid SYLK file dimension bounds** [120]
The SYLK format file you are importing from contains data items that are outside the bounds of the file.

**Invalid SYLK file format** [121]
The file you are trying to import from is not a SYLK format file.

**Invalid SYLK file header** [119]
The file you are trying to import from does not have the correct SYLK file header.

**Invalid TAG name** [284]
This message means that the name you have specified for the tag name is not in the .mdx file.

**Label field invalid**
The file being loaded does not have a valid label file format.

**Label file invalid** [54]
You ran or modified a label (.lbl) file not created with the CREATE/MODIFY LABEL commands.

**Left margin and indent must be less than right margin**
This is a Control Center word-wrap editor error. This message occurs when the indent is placed after the right margin, the left margin is placed after the right margin, the right margin is placed before the indent, or the right margin is placed before the left margin.

**Left margin plus indentation must be less than right margin** [221]
When specifying printer output, the sum of the left margin and paragraph indent you used exceeded the right margin.

**Line exceeds maximum of 1024 characters** **[18]**
A command line may not exceed 1,024 characters.

**Line number must be between 0 and page length** **[222]**
Range checking must be done on assignments of line number, and the line number must fit within the page length memory variable.

**Link is currently pending**
You selected **Create link** and then selected it again without first closing out the first pending link (query design).

**Lock table is full** **[217]**
You used up all the available record locks. The limit is 50.

**Log file corrupted** **[195]**
The transaction log file kept by dBASE IV is unreadable. Issue an END TRANSACTION command to clear file headers of ongoing transaction tags. You cannot do a ROLLBACK without a transaction log file. You must restore the file to its pre-transaction status from a backup copy.

**Log file not found** **[191]**
The transaction log file kept by dBASE IV is lost. Issue an END TRANSACTION command to clear the file headers of ongoing transaction tags.

**Log record does not match database record** **[196]**
The transaction log file does not match the database. Issue an END TRANSACTION command to clear the file headers of ongoing transaction tags.

**LOG(): Zero or negative** **[58]**
The argument of the natural log function may not be zero or a negative number.

**LOG10(): Zero or negative** **[292]**
The argument for a base 10 logarithm may not be zero or a negative number.

**Macro library doesn't exist** **[354]**
The macro library you are trying to call up is not in your current directory, or it does not exist. Check the library name and location and try again.

**Macro not found in the current library** **[355]**
The macro name or macro key code you used is not defined in the macro library you have loaded. If you create a new macro while a library is active in memory, you will write over duplicate macro names. Make sure you are defining a unique macro name or key code.

**Macro recording is in process** **[357]**
This is not an error message but displays while you are recording a macro on the message line.

**Macros are not allowed in user defined functions** **[393]**
User-defined functions may not contain program macros (the & function macros).

**Macros cannot expand flow-of-control commands** **[388]**
You cannot include a program macro (the & function) in control structures such as IF or SCAN.

**Macros nested too deep** **[361]**
You get this message if you attempt to nest more than 16 macros.

**Maximum field width exceeded**
This is a Control Center error. You see this message when you try to size a field to greater than the current maximum width of the layout.

**Maximum number of GET commands exceeded** **[310]**
Reduce the number of GET commands on one page.

**Maximum number of fields reached** **[296]**
The maximum number of fields per database file is 255.

**Maximum number of index tags already reached**
You tried to add a 49th index to the .mdx file by setting more than 48 index fields in database design to Y.

**Maximum number of tab stops exceeded**
This message comes from the Control Center word-wrap editor when you try to place more than 30 tab stops on the ruler line.

**Maximum popup nesting exceeded** **[182]**
You have exceeded the maximum pop-up menu nesting of 40.

**Maximum print width of 255 characters exceeded**
This occurs when label width times number of labels exceeds 255 characters.

**Maximum record length exceeded** **[137]**
The maximum record length is 4,001 bytes.

**Maximum record length exceeded in WK1 file** **[392]**
Your Lotus worksheet exceeds the maximum record length of 4,001 bytes.

**MDX file doesn't match database** **[207]**
The index file you want to use does not correspond to the database currently in use.

**MDX file full** **[203]**
The maximum number of .mdx files you can open at one time is 47.

**Memo fields cannot be prompted** **[280]**
You have attempted to define a POPUP window and to use a memo field as the PROMPT FIELD <field name>.

**Memory Error**
This is a DOS error message indicating that you have bad RAM. Quit dBASE IV and run your Hardware Diagnostics program from the disk that came with your computer to identify the location of the defective RAM chip(s). If you are not familiar with computer hardware, contact a service technician to diagnose and replace the bad memory chips.

**Memory variable already defined—cannot make PUBLIC** **[271]**
If you define a memory variable as private and then attempt to declare it as public, this message is displayed. You can declare a memory variable as public when you are defining it, but not afterwards.

**Memory variable cannot be defined here as PRIVATE** **[270]**
You attempted to do a PRIVATE command on a memory variable that has already been defined at the current level.

**Memory variable file invalid** **[55]**
You attempted to restore from a .mem file, but dBASE IV detected an error in the data type of one of the variables.

**Memos are not available** **[172]**
This happens when you open a .dbf that has an associated .dbt file, but the .dbt file cannot be found. dBASE IV prompts to ask whether you want to erase all memo information. If you respond with N, then dBASE IV sets a flag to avoid all memo operations. If you then attempt to access a memo field, you see this message.

**MENU has not been activated** **[178]**
You must activate a menu before you can select from it.

**MENU has not been defined** **[168]**
You must define a menu before you can activate it.

**Menu is already in use** **[181]**
You have attempted to activate a MENU that is already activated. A MENU cannot be activated twice at the same time.

**Minimum field width reached**
This is a Control Center error. You see this message when you try to size a field to smaller than 1.

**Missing END PRINTJOB for previous PRINTJOB** **[300]**
You are attempting to issue a PRINTJOB command while there is an unterminated PRINTJOB in the queue or in progress. Issue an END PRINTJOB command before starting a new PRINTJOB. Printjobs cannot be nested.

**Missing ENDTEXT for previous TEXT** **[341]**
You get this error if a TEXT command does not have a following ENDTEXT command in a procedure.

**Missing END TRANSACTION for previous BEGIN TRANSACTION** **[301]**
Each BEGIN TRANSACTION command must be terminated by an END TRANSACTION command. Transactions cannot be nested.

**Missing ENDCASE for previous DO CASE command** **[247]**
Each DO CASE command must have a terminating ENDCASE command.

**Missing ENDDO for previous DO WHILE command** **[246]**
Each DO WHILE command must have a terminating ENDDO command.

**Missing ENDIF for previous IF command** **[244]**
Each IF statement must have a terminating ENDIF statement.

**Missing ENDIF for previous IF/ELSE commands** **[245]**
Each IF/ELSE condition must have a terminating ENDIF statement.

**Missing ENDSCAN for previous SCAN command** **[269]**
A SCAN command must be terminated by an ENDSCAN.

**Missing EXPRESSION** **[152]**
An expression is missing from the syntax of a command such as DO WHILE, or from an IF statement.

**Missing index for ARRAY reference** **[299]**
You get this error message if you specify an array name without any indexes. For example, if you declare an array xxx, set one of its elements to a value, and then try to display the array element value without specifying its index, you will get this error message.

**Modify group only modifies group bands**
This occurs when a user highlights a non-group band and then selects **Modify group** from the **Bands** menu.

**More than one sort option in a column** **[325]**
During a semantic check, dBASE IV finds a column that contains more than one sort directive (query design).

**Must be a valid dBASE expression. Press any key to continue**
The field renamed is not a valid dBASE field name. This also occurs when a user tries to add an unnamed calculated field to the view.

**NDX index limit reached** **[204]**
You can have ten .ndx index files open. Close some indexes before you attempt to open more index files.

**NDX index may not be DESCENDING** **[235]**
Use an .mdx index for descending order indexing.

**Network server busy** **[148]**
Your network server is busy. Please wait until the network server can handle your request. See your network administrator.

**No bars have been defined for this popup** **[166]**
Each pop-up menu must have bars defined for it.

**No database in USE** **[52]**
You specified a command that requires an open database file. Open a database file with the USE command.

**No database in use. Form contains the following fields:**
This is displayed in the **Forms** menu when the form being modified references a database that is not in use. The list of fields used by the form is displayed.

**No DBF files are present**
The Control Center displays this message when there are no .dbf files in a file pick list.

**No fields of the requested type are present. Press any key to continue**
You are requesting the **Fields** menu when there are no fields of the required type in either the file structure or the SET FIELDS list.

**No fields to process** **[47]**
You attempted to use a command such as BROWSE or EDIT with a database file that has no fields in the SET FIELDS list.

**No fields were found to copy** **[138]**
You tried to copy fields, but did not define a fields list first.

**No files of the requested type are cataloged**
You selected the **Format** or **Index** options from the full-screen **SET** menu and there are no files in the current catalog with the proper file extensions to display in the filename list.

**No files of the requested type are cataloged. Press any key to continue**
The catalog that is open does not contain any files of the type you are looking for.

**No files of the requested type are present. Press any key to continue**
You selected the **Format** or **Index** options from the full-screen **SET** menu and there are no files in the current directoy with the proper file extensions.

**No files of the requested type in this drive or catalog** **[53]**
You are doing a directory search with wildcards for a file type from DOS, and there are no files that match the filter.

**No format files in current directory**
You cannot select the **Format** option in CREATE/MODIFY VIEW when there are no format files in the current directory.

**No format files in the catalog for the current database**
The catalog of the database currently in use contains no format files. You cannot choose the Format option.

**No index files for current database in the file catalog**
Create an index file with SET CATALOG ON for the database you are using.

**No label forms in catalog file for current database**
Create a label file with SET CATALOG ON for the database from which you want to print labels.

**No more windows available** **[213]**
You may have a maximum of 20 windows.

**No previous BEGIN TRANSACTION to match this command** **[303]**
You have issued an END TRANSACTION command, but there is no currently active BEGIN TRANSACTION command in progress to end.

**No previous DO CASE to match this command** **[250]**
Each ENDCASE must have a preceding DO CASE command. Check your program and correct unbalanced syntax.

**No previous DO WHILE to match this command** **[251]**
Each ENDDO command must have a preceding DO WHILE command.

**No previous DO WHILE/SCAN/PRINTJOB to match this cmd** **[304]**
You issued an ENDDO/ENDSCAN/END PRINTJOB command when there is no unterminated DO/SCAN/PRINTJOB command that is pending.

**No previous IF to match this command** **[249]**
Each ENDIF command must have a preceding IF statement.

**No previous PRINTJOB to match this command** **[305]**
You have issued an END PRINTJOB command, but there is no PRINTJOB command to terminate.

**No previous SCAN to match this command** **[302]**
You have issued an ENDSCAN command, but there is no previous SCAN command to match this one.

**No printjob is in progress** **[282]**
You are trying to issue an END PRINTJOB command when there is no current PRINTJOB to terminate.

**No program file found for this application** **[334]**
The .prg file for the application you are trying to run is not in your current directory or path statement. Locate the program file on your disk and use the correct file specification.

**No records selected** **[365]**
This message is used in BROWSE and indicates that there are no records to browse. This can occur if a filter (SET FILTER TO) is used and there are no selected records to BROWSE.

**No report forms in catalog file for current database**
The catalog associated with the database in use does not contain report form files. Create report forms with SET CATALOG ON.

**No search string specified**
You selected **Forward/Backward search** from the **GoTo** menu or pressed **Shift-F5** before entering a search string.

**No selection made for MOVE**
Press **F6** to select a field to move before you press **F7**.

**No view files in the catalog for current database**
CREATE VIEW for the current database with SET CATALOG ON.

**Not a character expression** **[45]**
You did not provide a character expression for a field that requires one (for example, trying to set delimiters to a numeric expression).

**Not a dBASE database** **[15]**
You are trying to use a non-dBASE format file with dBASE IV. Check the file format. If it is a dBASE II file, you must rename its extension from .dbf to .db2 for dBASE IV to recognize it.

**Not a DBO file: <filename>** **[253]**
The filename you used following a DO command is not a compiled program file even if it has a .dbo extension. Check your directories to find the file you want. Recompile from the .prg file.

**Not a defined error**
This message catches any untrapped DOS error messages. dBASE IV currently traps about ten DOS error messages. If a DOS error occurs that dBASE IV does not trap (an extremely rare situation), then this message gets displayed.

**Not a logical expression** **[37]**
This program or procedure requires a logical true (.T.) or false (.F.), such as in a DO/WHILE clause.

**Not a numeric expression** **[27]**
You are attempting to use a non-numeric expression with a mathematical, statistical, or trigonometric function, or with a command that takes a numeric expression as an argument.

**Not a valid dBASE II database** **[257]**
The database you are attempting to import is not a valid dBASE II database, or you did not change its extension to .db2.

**Not a valid expression** **[233]**
You have used an invalid dBASE IV expression in a program. Check the syntax in this manual.

**Not a valid Framework II database/spreadsheet** **[256]**
The file you are trying to import is not a Framework II format file.

**Not a valid PFS file** **[140]**
The file you are trying to import is not a valid PFS:FILE file.

**Not a valid QUERY file** **[134]**
You attempted to SET FILTER TO an invalid query file. Create a new query file.

**Not a valid RapidFile database** **[255]**
The file you are trying to import is not a valid RapidFile file.

**Not a valid VIEW file** **[127]**
You are trying to SET VIEW TO or MODIFY VIEW to a file that may be damaged. Repeat CREATE VIEW FROM ENVIRONMENT or CREATE VIEW to get a new view file.

**Not a valid window file** **[358]**
The window definition file that you tried to activate is not the name of a window file. Check your file name and retry.

**Not an array** **[229]**
You tried to use a memory variable as an array when it was not declared as one.

**Not enough disk space for operation** **[275]**
You have run out of disk space for writing the output of a procedure or an INDEX or SORT. Make room by removing files from the hard disk, or specify another drive. If you were sending an output file to a floppy and the file exceeded the capacity of the floppy disk, you may not output to a floppy.

**Not enough disk space for SORT**
You need enough disk space to store two files the same size as the source file. Direct the sort output file to another drive, or make room on the disk by removing files.

**Not enough records to sort** **[277]**
You cannot sort fewer than two records.

**Not suspended** **[101]**
You issued a RESUME command when the file execution was not suspended.

**Numbers are not allowed in the CURRENCY symbol** **[295]**
Use only characters to define a currency symbol.

**Numeric overflow (data was lost)** **[39]**
You exceeded the numeric limits of the filed or memory variable as a result of a mathematical operation.

**ON PAD already defined for this prompt pad** **[170]**
You have previously defined an ON SELECTION PAD for the prompt pad you named in your last command. You may not use more than one ON SELECTION PAD for one pad.

**ON SELECTION already defined for this prompt pad** **[169]**
ON PAD has already been issued for the pad for which you are issuing an ON SELECTION PAD command.

**Only one update operation allowed per query** **[328]**
When you leave the pothandle, dBASE IV detects that more than one update operator (replace, append, delete, untag) has been specified (query design).

**Only windows, boxes, and fields can be sized**
This is a Control Center error. You see this message if you press **Shift-F7** on an object that cannot be sized.

**Operation not allowed for calculated fields** **[371]**
You cannot SORT or INDEX on a calculated field.

**Operation with Logical field invalid** **[90]**
You are trying to SORT or INDEX on a logical field.

**Operation with Memo field invalid** **[34]**
You are trying to SORT or INDEX on a memo field.

**Operation not allowed in the calculated field skeleton**
You attempted to make an entry in the pothandle of a calculated field skeleton.

**ORDER must be specified by index TAG or filename** **[329]**
You have specified a number with the SET ORDER TO command, and only an .mdx file is in use. You must specify an index TAG name. You can specify a number only if you have .ndx files.

**ORDER TAG not found** **[208]**
This message indicates that the SET ORDER TO command was used with an .mdx file TAG that does not exist. Verify the TAG names in the .mdx file that is in use. Then reissue the SET ORDER TO command.

**Out of memory variable memory** **[21]**
You do not have enough system memory to define more memory variables.

**Out of memory variable slots** **[22]**
You do not have enough system memory to define more memory variables.

**PAD has not been defined** **[164]**
A pad must be defined for popups and menus.

**PAD has not been defined for this menu** **[180]**
You are trying to activate a menu that has been defined but does not contain any pads. A pad must be defined for a menu before you can activate it. A menu that has only a name cannot be displayed.

**PARAMETERS command must be at top of procedure** **[243]**
You must have the PARAMETERS command as the first statement in any compiled procedure. Edit your program file and place the PARAMETERS command at the top of the file.

**Password and confirmation mismatch** **[154]**
The first password you typed is different from the second word you typed. Type your password again.

**Password file is in use by another** [349]
The password file is being used by another user. Try again later or see your system administrator.

**Password has not been defined** [157]
You do not have a password to use with the PROTECT command. See your system administrator.

**PFS does not allow row numbers higher than 20**
You cannot export this file to PFS:FILE without reducing the number of rows.

**Place fields in the VIEW skeleton first** [321]
You pressed **F2** on a view query, and there are no fields in the view skeleton.

**Please put a database file or view into use first** [281]
You must put a file into USE before you can perform any commands on it.

**Please put view or dbf in use first**
You attempted to create an object, and there is not a database file or view in use.

**POPUP has not been activated** [179]
You must activate a pop-up menu before you can make selections from its pads.

**POPUP has not been DEFINEd** [165]
You must define a pop-up menu before you can define pads for it or activate it.

**Popup is already in use** [182]
You have attempted to activate a POPUP that is already activated. A POPUP cannot be activated twice at the same time.

**POPUP is too small** [287]
The pop-up window is too small to have any selection bars. This is illegal. Define a larger popup and selection bars.

**Position out of window** [288]
The coordinates you are specifying for a pad or a prompt are outside the window boundaries.

**Printer is either not connected or turned off** [126]
You are trying to send output to a printer that is not connected to the port you specified, or it is not on-line, or it is turned off. Check the printer.

**Printer not ready** [125]
You are attempting to send output to a printer which is not on-line, or may be turned off. Check the printer.

**Printjobs cannot be nested** [337]
Each PRINTJOB command must be terminated by an END PRINTJOB command before you can start another printjob. Issue an END PRINTJOB command to clear this error.

**PROCEDURE command is required** [242]
This occurs during the compiling of a procedure file. After an unconditional return command is located (for example, one that is not located within an IF or DO case), the very next command must be the PROCEDURE command to indicate the beginning of the next procedure.

**Procedure is too large ( > 64K)** **[258]**
A compiled procedure is limited to exactly 65,520. Your procedure is too large. Divide the procedure into smaller functional groups.

**Procedure not found: < procedure name >** **[252]**
The procedure you named to DO is not part of the database file in use.

**PROCEDUREs cannot return a value** **[385]**
Procedures cannot return a value. Use a dBASE IV or user-defined function to get a value back.

**Production MDX file is damaged** **[289]**
The production .mdx file does not agree with its corresponding database file. Reindex the database file to correct this error.

**Production MDX file not found** **[210]**
The .mdx file associated with the database file you are using cannot be found. dBASE IV asks you if you want to continue.

**PROMPTS for this popup have already been defined** **[279]**
You have used the PROMPT option to define the contents of the popup. You may not use the BAR option if you have used the PROMPT option.

**Query not executable. Saved anyway**
You saved an incorrectly defined query.

**Query not valid for this environment** **[143]**
This is a query file error; it gets generated when a field referenced by the query file is not in any of the files in use.

**Query too complex** **[338]**
This message occurs if the total size of the fields in a unique query is greater than 100, or if the command line generated by the query engine is greater than 1,024 characters.

**Quick Report is not available while a form, report, or label is being modified** **[351]**
You cannot use the Quick Report form option while modifying a form, report, or a label.

**Read error** **[254]**
This is an operating system error indicating a problem with a read operation.

**Read error on < drive >**
This message contains the name of the drive where the read error has occurred. This is an operating system error indicating a problem with a read operation.

**READ or @ command cannot follow FMT termination commands** **[359]**
A READ or an @ command has been detected after the termination section of a format (.fmt) file. These commands are not allowed after the termination of a format file.

**Record in use by < username >** **[373]**
In multi-user operation, the record you want to use is locked by the user identified in the error message.

**Record in use by another** **[109]**
The file you want to use is currently in use on the network by another user.

**Record not in index** **[20]**
You are trying to go to a record that is not in the index. Create a new index for this file, or do not use an index file.

**Record not inserted** **[25]**
You attempted an INSERT command, but the record did not get inserted. Retry inserting the record.

**Record out of range** **[5]**
You are trying to go to a record that does not exist.

**Records do not balance (program error)**
This is an internal dBASE IV message that should never appear. There is a problem with the sort algorithm of dBASE IV, and the sort output is corrupted.

**Recursive keystroke macro call: < macro name >** **[363]**
When you are recording keystrokes for a given key and you press that key, you see this error message and the key is not recorded. If it were saved, it could cause a looping operation. The macro name is displayed after the colon.

**Relation record in use by another** **[142]**
The active database file has a SET RELATION TO another file that is in use by another.

**Remove group only removes group bands**
This occurs when you highlight a non-group band and then select **Remove group** from the **Bands** menu.

**REPLICATE( ): String too large** **[88]**
The string you are attempting to replicate exceeds 255 characters.

**Report field invalid**
The report file being loaded does not have a valid format.

**Report file invalid** **[50]**
The report file you are trying to use does not have a valid report file format. Check your filename or CREATE a new report form.

**Restricted command: not allowed in this context** **[389]**
This command may not be used in a user defined function or with ON COMMAND.

**RETURN TO is invalid in user defined function** **[383]**
The RETURN command returns a value and terminates all user-defined functions. You cannot RETURN TO a different program or procedure from a user-defined function.

**Right margin must be less than or equal to 255** **[225]**
The right margin setting cannot be greater than the page width.

**Rollback database cannot be executed inside a transaction** **[197]**
A ROLLBACK command with a database filename can only be used after a hardware failure. If you issue a ROLLBACK command with a filename while a transaction is still in progress, you get this message.

**Screen file invalid**
The form file being loaded does not have a valid format.

**Screen file invalid** [283]
You have modified a format (.fmt) file without changing the corresponding screen (.scr) file. The screen file does not match the .fmt file.

**Select box with F6 only**
This is a Control Center error. You see this message when you try to use navigation keys during an extended selection of a box.

**Sort order type must be A or D**
You pressed the wrong key when selecting a sort-order type **Sort on a field list** in database design.

**Source does not correspond to the object** [95]
You get this error if a program is being displayed when SET ECHO is ON, or code is displayed in the debugger code window and the source does not correspond to the object code. This can occur if a .prg file has been modified and has not been recompiled. Under these circumstances, the displayed source code does not correspond to the object code being executed.

**SPACE( ): Negative** [60]
You used a negative number for the SPACE() function.

**SPACE( ): Too large** [59]
You used a number greater than 254 for the SPACE() function.

**SQL run-time error** [335]
This message is displayed if an error occurs during the execution of an SQL statement.

**SQL System Tables may not be USEd** [218]
You have attempted to USE an SQL System Table with the dBASE USE command. The USE command may not be used with SQL tables.

**SQRT( ): Negative** [61]
The square root function cannot have a negative number as its input.

**STORE: String too large** [79]
Strings are limited to 254.

**STR( ): Out of range** [63]
This is generated when the third argument exceeds the second argument in the STR() function.

**String not found**
A search operation failed to find the specified string.

**Structure invalid** [33]
The structure defined in the structure-extended file that you used with the CREATE FROM command is invalid.

**STUFF( ): String too large** [102]
The string you are trying to insert into a field is too large to fit into that field.

**SUBSTR( ): Start point out of range** [62]
This message occurs when the second argument is greater than the length of the input string. You cannot print a character that is not contained within a string.

**Syntax error** [10]
Syntax error in query design.

**Syntax error in contents expression**
This error is displayed in the label printer. It means that the dBASE III PLUS label form contains an illegal expression.

**Syntax error in query definition**
This message appears when you load a query that has a syntax error, such as mismatched delimiters, or a missing comma.

**Tab stops must be in ascending order** **[226]**
The values assigned to the memory variable *_tabs* must be given in ascending order.

**Tabstop value exceeds 255** **[399]**
You cannot have a Word Wrap editor tab stop value that is greater than 255.

**Table full** **[105]**
You can load a maximum of 16 binary program files into the CALL table which you can then call from within dBASE IV. Reduce the number of files.

**TAG not found** **[209]**
This error message is generated if you specify a tag which cannot be found in the open .mdx file.

**Target file not found in catalog** **[343]**
If you are running a query update from the Control Center and the .upd file cannot be found in the catalog, this error message is displayed.

**There are no files of the type requested in the current directory or catalog** **[53]**
You are searching the current directory or catalog from the Control Center with a wildcard filter. There are no files of the type you specified, in either the directory or the catalog.

**This file cannot be modified** **[342]**
This message indicates that the file on which the cursor is positioned in the Contol Center cannot be modified when design key (**Shift-F2**) is pressed. Files with file extensions of .qbo, .upo, .fmo, .fro, .lbo, .dbo, .exe, and .com cannot be modified.

**Too many @ commands on one page of format file**
This message is used with PFS:FILE export and indicates that there are too many @ commands to export to PFS:FILE form. You are attempting to use more than 200 @ commands on one page. Reduce the number of @ commands.

**Too many fields in WK1 file** **[391]**
The Lotus worksheet you are trying to import exceeds the number of allowed fields.

**Too many files are open** **[6]**
You are trying to exceed the number of files that you set in the Config.sys file with the command FILES = . Use fewer files, or close all files and quit dBASE IV to increase the number in the FILES = line in Config.sys. Reboot after changing the Config.sys file for the new value to take effect.

**Too many indexes** **[28]**
You may have a maximum of 10 .ndx files and 47 .mdx files.

**Too many merge steps** **[278]**
This message involves an error in sorting a very large file. The final phase of the sort algorithm merges increasingly large blocks of sorted records. This message is generated when the 16th merge step is reached and the sorted records are still not completely merged. It would take an immense file to reach this limit.

**Too many pages in format file**
This message is used with PFS:FILE export and indicates that there are too many format file pages to export to PFS:FILE form.

You are allowed a maximum of 32 pages in a format file. Divide the file into two or more format files for more pages.

**Too many sort key fields** **[276]**
You may use ten sort key fields.

**Too many WITH operators** **[319]**
When performing a semantic check, dBASE IV finds too many WITH operators (used with the REPLACE operator) in the file skeleton (query design).

**Total label width exceeds maximum size**
This message occurs during label generation using dBASE III PLUS type labels.

The maximum allowable label width is 250 columns.

**Total size of fields in view too large for UNIQUE query** **[381]**
This message occurs if the total size of the fields in a unique query is greater than 100.

**TRAP can not be turned OFF while Debugger is active** **[390]**
You must SET TRAP ON or OFF before you call the debugger program.

**Unable to load Command.com** **[92]**
You attempted to go to DOS from the Control Center, but there is not enough RAM to do this. Close some files or remove memory variables, and try again.

**Unable to load file**
This message occurs when a user attempts to load a file that does not correspond to any of the formats (report, form, and label files) handled by the layout editor. This can also occur if the file is corrupted.

**Unable to LOCK** **[129]**
The record you want to lock is already in use by another user.

**Unable to SKIP** **[128]**
The record you want to SKIP to is in use by another user. You may access any other unlocked record, or wait until that record is unlocked.

**Unassigned file number** **[2]**
This message can occur if you attempt an I/O operation by using a file handle number that hasn't been opened. This is an internal error and should not occur under normal dBASE IV usage.

**Unauthorized access level** **[133]**
You are trying to access a file which you are not authorized to access. Please see your system administrator about access levels.

**Unauthorized login** **[132]**
You are trying to log in without a valid password. See your system administrator for a valid password assignment.

**Unbalanced parenthesis** **[8]**
Each function argument must have an opening and a closing parenthesis.

**Undefined box border** **[228]**
This error occurs when you define box borders for printing and you omit the border definition or provide invalid border definitions.

**Unknown function key** **[104]**
You have not defined the function key or key combination you are trying to use.

**Unknown system memory variable** **[220]**
You have a memory variable beginning with an underscore character that is not one of the dBASE IV system memory variables. Underscore is reserved for system memory variables. Please refer to Chapter 5 or *Quick Reference* for information on the system memory variables.

**Unrecognized phrase/keyword in command** **[36]**
This can occur with any command that has keywords or phrases. It indicates an illegal keyword in a command line.

**Unterminated string** **[35]**
A string must be terminated with a legal dBASE IV delimiter. These are single quotes, double quotes, or square brackets. Check the syntax for the specific command.

**Unterminated transaction file exists, cannot start new transaction** **[183]**
You have an unterminated transaction in progress. Issue an END TRANSACTION command to clear the integrity tag from the file header before you can start a new transaction.

**User defined functions must return a value** **[382]**
All user-defined functions must return a value. Change the user-defined function to return a numeric value.

**User name has not been defined** **[156]**
The PROTECT utility does not contain the name you are trying to use. See your system administrator.

**Variable not found** **[12]**
The variable you specified is not a field name or does not exist in the currently selected work area or the work area specified by ALIAS. Also, the memo file you are trying to copy or append to or from does not exist.

In query design, dBASE IV found an example variable that was not defined.

**View cannot hold any more fields (255 already)** **[353]**
You tried to add a 256th field to the view.

**WARNING: File definition has changed** **[320]**
The structure of one of the files has been modified since you last modified the query.

**Warning: key expression uses ALIAS or MEMVAR** [387]
This message warns you that if the dBASE IV environment changes, an index key expression which uses an ALIAS or a memory variable may become invalid.

**WARNING on line <program line number> <warning message>** [315]
This is a compiler warning message header. It indicates that a line of code contains some unsupported command syntax or erroneous characters at the end of the line. Compilation continues and an object (.dbo) file is generated.

**WINDOW coordinate(s) outside of allowable screen space** [332]
You are trying to define a window that is not within the coordinates of 0,0 and 22,79.

**WINDOW has not been defined** [214]
You are attempting to activate a window that has not been defined. Define the window first.

**WINDOW is too small** [285]
You have defined a window that is too small to be used by BROWSE. The window must be four rows high, and large enough to contain at least one field.

**WINDOW not presently on the screen** [215]
You issued a DEACTIVATE WINDOW <window name>, and the window you named is not active on the screen.

**Work area already used in a relation** [298]
A relation chain can only refer to a database file in a work area once. You cannot define a database file twice in a relation chain. If you do, you get this message.

**Work area reserved by SQL** [219]
You cannot use a file in a work area if it has already be reserved for use by SQL.

**Write error on <drive>**
This is an operating system error indicating a write error on the drive identified in the message.

**Wrong number of parameters** [94]
You are using a DO...WITH construct in calling a program file with a .prg extension. The program file has the keyword PARAMETERS followed by an incorrect number of parameters in it.

**Zoom window command editor, CTRL-END exits to dot prompt**
This is an information message from the zoom window command editor.

**** Not Found **** [82]
This message occurs in Browse or Edit in a full-screen operation when a field cannot be found.

**** WARNING ** Data will probably be lost. Confirm? (Y/N)** [70]
This message occurs when a disk is full. If you respond with Y, you will be allowed to go through and delete files from the full directory to make space for the file you are working with. If you respond with N, you will lose the data you are working with when you exit dBASE IV.

**** WARNING ** Indexes could not be updated [378]**
dBASE IV was not able to update the index files associated with the database in use, because it could not find the index files or these indexes were not in USE.

**** WARNING ** Uncompleted transaction found [190]**
You are attempting to use the BEGIN TRANSACTION command before first issuing an END TRANSACTION command.

***** Unrecognized command verb [16]**
You are using a command that is not part of dBASE IV syntax.

*****Execution error on [80]**
You have supplied a bad argument to a function. The name of the function that you were trying to use is displayed after this message. Example:

> *****Execution error on SUBSTR(): Start point out of range**

**– : Concatenated string too large [76]**

**+ : Concatenated string too large [77]**
The string resulting from the concatenation of two strings exceeds 254 characters. The difference between the two messages is in the concatenation operator that was used to create the large string: either a + or a –.

**, Database in Use: [71]**
This is a status message that appears with the display status command. The entire message is:

> **Select area: 1, Database in use: filename.dbf Alias: Filename**

**^ or **: Negative base, fractional exponent [78]**
You cannot use a negative number as a parameter with arithmetic and trigonometric functions. Exponents must be integers.

**^—- Keyword not found [86]**
A keyword you used in the Config.db file is not a valid dBASE keyword. See Chapter 6, "Customizing dBASE IV," for a list of valid keywords.

**^—- Out of range [75]**
The parameter you are using with this function exceeds the allowed range. Check Chapter 4 "Functions," for the function range.

**^—- Truncated [74]**
A line of text in the Config.db file was too long or unterminated, and was truncated. The maximum allowed string length for Config.db is 255.

**^^ Expected ON or OFF [73]**
The SET command you are changing can only accept ON or OFF as its parameter setting.

# dBASE IV Specifications

This appendix outlines the specifications for dBASE IV files and operations.

## Database File

Number of records: 1 billion
Number of bytes: 2 billion
Record size, in .dbf: 4,000 bytes
Number of fields: 255

## Index File

Number of indexes per multiple index file: 47
Blocksize: 16,384 bytes (default, 2,048 bytes)

## Field Sizes

Character fields: 254 bytes
Date fields: 8 bytes
Logical fields: 1 byte
Type N fields: 20 digits
Type F fields: 20 digits
Characters in a field name: 10

## Arrays

Dimensions: 2
Total size (rows x columns): 1170

## Multi-user Procedures

Locks — Maximum number of locks: 50 (files and records)
Reprocess — Maximum number of counts: 32,000
Refresh — Maximum number of seconds: 3,600

## File Operations

(Note that the number of files you can open simultaneously may be limited by settings in the Config.sys and Config.db files, and by available memory.)

Open files of all types: 99
Open database files: 10
Open memo files per active database: 1
Open index files per active database: 10
Open format files per active database: 1
Open procedure files per run: 1

## Numeric Accuracy

(Note that the decimal point does not count as a digit in determining accuracy.)

Type F numbers:

15.9 digits
Largest number: 0.9 x E + 308
Smallest number: 0.1 x E − 307

Type N numbers:

10 to 20 digits based on the setting of SET PRECISION TO <expN>
Largest number: 0.9 x E + 308
Smallest number: 0.1 x E − 307

## Memory Variables

(Note that available memory may limit the maximum.)

Defaults to 500 variables
Can be changed in Config.db to a maximum of 15,000 variables

## Run-time Symbols

(Note that available memory may limit the maximum.)

Defaults to 500
Can be changed in Config.db to a maximum of 15,000

## Compile-time Symbols

(Note that available memory may limit the maximum.)

Defaults to 500
Can be changed in Config.db to a maximum of 5,000

## Capacities

The capacities for the various dBASE IV operations, such as editing and report writing, are listed below.

(Note that available memory may limit these capacities.)

### Word Wrap Editor

Number of lines: 32,000
Line length:

| | |
|---|---|
| MODIFY COMMAND (.PRG) | 1,024 |
| MODIFY FILE | 1,024 |
| MEMO FIELDS | 1,024 |
| SQL (F9) | 1,024 |
| HISTORY | 1,024 |
| REPORT | 255 |

### Forms

Number of rows: 32,767
Width: 80 characters

### Reports

Width: 255 characters
Number of databases: 9
Number of indexes per database: 10
Number of relations: 9
Number of reports per database: unlimited
Number of pages: 32,767
Number of nested group bands: 44
Number of fields: limited by memory
Number of copies: 32,767

### Labels

Width: 255 characters
Length: 255 lines
Number of labels across: 15
Number of copies: 32,767

### QBE

Joined files: 8

## SQL

Tables in a join: limited by available memory
Number of cursors: 10
Number of indexes per table: 47
SQL statement length: 1,024 characters

## Applications Generator

Number of objects on work surface: depends on complexity of objects and available memory
Size of editor: 4K

## Miscellaneous Capacities

Command line length: 255 bytes at dot prompt, 1,024 bytes in edit window
Maximum nesting level of control commands: set by DO parameter in Config.db
Procedures per program/procedure file: 963
Procedure size limit: 65,520 bytes of code
Maximum number of active procedures: limited by memory
Sort levels sorted simultaneously: 16
Maximum number of GET commands in a format file: 2,000
Printed page length: 32,767 lines
Number of macros: 35
Number of binary files that can be loaded: 16
Number of printer drivers: 4 configured
Number of fonts: 5 per printer driver
Number of work areas: 10
Number of programmable function keys: 29
Number of recursive calls: limited by available memory and virtual stack size

**NOTE**
*An* active procedure *is one that can be reached by a RETURN, or is contained in the current procedure file or program file.*

# Sample Files

This appendix contains descriptions of the example database files and their associated index files used in *Language Reference*. These files are also on disk.

Several examples in *Language Reference* are excerpts from Menus.prg, a program which demonstrates horizontal bar and pop-up menus. A complete listing of Menus.prg is included in this appendix.

## Client.dbf

```
Structure for database: CLIENT.DBF
Number of data records:       8
Date of last update   : 11/05/87
Field  Field Name  Type       Width    Dec    Index
    1  CLIENT_ID   Character      6               Y
    2  CLIENT      Character     30               Y
    3  LASTNAME    Character     15               N
    4  FIRSTNAME   Character     15               N
    5  ADDRESS     Character     30               N
    6  CITY        Character     20               N
    7  STATE       Character      2               N
    8  ZIP         Character     10               N
    9  PHONE       Character     13               N
   10  CLIEN_HIST  Memo          10               N
** Total **                     152

Record#  CLIENT_ID CLIENT                          LASTNAME        FIRSTNAME
      1  A00001    WRIGHT & SONS, LTD              Wright          Fred
      2  L00001    BAILEY & BAILEY                 Bailey          Sandra
      3  C00001    L. G. BLUM & ASSOCIATES         Martinez        Ric
      4  L00002    SAWYER LONGFELLOWS              Peters          Kimberly
      5  A00005    SMITH ASSOCIATES                Yamada          George J.
      6  C00002    TIMMONS & CASEY, LTD            Timmons         Gene
      7  B12000    VOLTAGE IMPORTS                 Beluga          Yuri
      8  A10025    PUBLIC EVENTS                   Beckman         Riener
```

Note that the following is a continuation of the above file.

```
ADDRESS                   CITY          STATE ZIP        PHONE          CLIEN_HIST
3232 48th St.             New York       NY   11101      (718)555-7474  MEMO
5132 Livingston Dr        Long Beach     CA   90803-0408 (213)555-1104  MEMO
4818 Allendale Ave        Santa Fe       NM   87501      (505)555-3232  MEMO
12300 N Elm St            Dallas         TX   75002      (214)555-5603  MEMO
7500 Santa Monica Blvd    Los Angeles    CA   90055-1319 (213)555-4300  MEMO
310-2090 Comex St         Vancouver      BC   V6G 1E8    (604)555-7644  MEMO
8506 Habana Ave           Tampa          FL   33614      (813)555-5522  MEMO
332 S. Michigan Ave       Pasadena       CA   91125-0001 (818)555-3842  MEMO
```

In the following section you will find listings of the information contained in the Memo file associated with Client.dbf:

```
CUSTOMER: Record No      1

85-200  08/02/85
    C-300-400 BOOK CASE              535.00 1

CUSTOMER: Record No      2

86-245  09/22/86
    C-700-2020 FILE CABINET,2 DRAWER  100.00 2
    C-700-4030 FILE CABINET,4 DRAWER  150.00 2
86-303  12/06/87
    C-222-1001 CHAIR, DESK           1750.00 1
87-109  03/09/87
    C-400-2060 TABLE, END             250.00 1
    C-500-6050 LAMP, FLOOR            165.00 1
87-112  03/20/87
    C-222-3020 CHAIR, SIDE            350.00 2

CUSTOMER: Record No      3

86-155  06/31/86
    C-600-5050 DESK, SECRETARY       1100.00 1
    C-222-3010 CHAIR, SIDE            500.00 1
    C-400-2080 TABLE, END             250.00 1
86-248  09/28/86
    C-600-5050 DESK, SECRETARY       1100.00 1
    C-222-3010 CHAIR, SIDE            500.00 1
    C-700-2020 FILE CABINET,2 DRAWER  100.00 1
    C-500-6050 LAMP, FLOOR           1100.00 1
86-312  12/17/86
    C-400-5000 TABLE, COFFEE          875.00 1
87-106  02/10/87
    C-111-8050 SOFA, 8-FOOT          1200.00 1
87-108  02/23/87
    C-222-1000 CHAIR, DESK           1250.00 1

CUSTOMER: Record No      4

No entries
```

```
CUSTOMER: Record No     5

85-187  06/07/85
    C-111-6045 SOFA, 8-FOOT           1325.00 1
87-113  03/24/87
    C-300-2020 BOOK CASE               125.00 1

CUSTOMER: Record No     6

87-107  02/12/87
    C-222-1000 CHAIR, DESK            1250.00 1
87-110  03/09/87
    C-700-2020 FILE CABINET,2 DRAWER    75.00 1
    C-700-4020 FILE CABINET,4 DRAWER   100.00 1

CUSTOMER: Record No     7

No entries

CUSTOMER: Record No     8

87-105  02/03/87
    C-111-6010 SOFA, 6-FOOT           1200.00 1
    C-111-6015 SOFA, 6-FOOT            650.00 1
```

The following is a list of the index keys used for the Client.dbf file:

```
Master Index file: CUS_NAME.NDX  Key: Lastname+Firstname
Production MDX file: CLIENT.MDX
          Index TAG: CLIENT  Key: CLIENT
          Index TAG: CLIENT_ID  Key: CLIENT_ID
```

## Transact.dbf

```
Structure for database: TRANSACT.DBF
Number of data records:      12
Date of last update   : 11/05/87
Field  Field Name  Type       Width    Dec    Index
    1  CLIENT_ID   Character      6               Y
    2  ORDER_ID    Character      6               Y
    3  DATE_TRANS  Date           8               N
    4  INVOICED    Logical        1               N
    5  TOTAL_BILL  Numeric        8      2        N
** Total **                      30
```

```
Record# CLIENT_ID ORDER_ID DATE_TRANS INVOICED TOTAL_BILL
      1 A10025    87-105   02/03/87   .T.         1850.00
      2 C00001    87-106   02/10/87   .T.         1200.00
      3 C00002    87-107   02/12/87   .T.         1250.00
      4 C00001    87-108   02/23/87   .T.         1250.00
      5 L00001    87-109   03/09/87   .T.          415.00
      6 C00002    87-110   03/09/87   .T.          175.00
      7 L00002    87-111   03/11/87   .F.         1000.00
      8 L00001    87-112   03/20/87   .T.          700.00
      9 A00005    87-113   03/24/87   .T.          125.00
     10 B12000    87-114   03/30/87   .F.          450.00
     11 C00001    87-115   04/01/87   .F.          165.00
     12 A10025    87-116   04/10/87   .F.         1500.00
```

The following is a list of the index keys used for the Transact.dbf file:

```
Production MDX file: TRANSACT.MDX
          Index TAG: CLIENT_ID  Key: CLIENT_ID
          Index TAG: ORDER_ID  Key: ORDER_ID
```

## Stock.dbf

```
Structure for database: STOCK.DBF
Number of data records:       17
Date of last update   : 11/05/87
Field  Field Name  Type       Width    Dec    Index
    1  ORDER_ID    Character      6               Y
    2  PART_ID     Character     10               N
    3  PART_NAME   Character     21               Y
    4  DESCRIPT    Character     30               N
    5  ITEM_COST   Numeric        8      2        N
    6  QTY         Numeric        3               N
** Total **                      79

Record# ORDER_ID PART_ID    PART_NAME             DESCRIPT                   ITEM_COST QTY
      1 87-105   C-111-6010 SOFA, 6-FOOT          LEATHER, BROWN, HIGHBACK     1200.00   1
      2 87-105   C-111-6015 SOFA, 6-FOOT          VELVET, GREY, FRENCH          650.00   1
      3 87-106   C-111-8050 SOFA, 8-FOOT          VELVET, BLUE, FRENCH         1200.00   1
      4 87-107   C-222-1000 CHAIR, DESK           LEATHER, BROWN, HIGHBACK     1250.00   1
      5 87-108   C-222-1000 CHAIR, DESK           LEATHER, BROWN, HIGHBACK     1250.00   1
      6 87-109   C-400-2060 TABLE, END            WOOD, OAK, 2-FOOT, SQUARE     250.00   1
      7 87-109   C-500-6050 LAMP, FLOOR           BRASS, 6-FOOT, ENGLISH        165.00   1
      8 87-110   C-700-2020 FILE CABINET,2 DRAWER METAL, BROWN                   75.00   1
      9 87-110   C-700-4020 FILE CABINET,4 DRAWER METAL, BROWN                  100.00   1
     10 87-111   C-222-1001 CHAIR, DESK           LEATHER, BROWN               1000.00   1
     11 87-112   C-222-3020 CHAIR, SIDE           PLASTIC, GREY                 350.00   2
     12 87-113   C-300-2020 BOOK CASE             WOOD, TEAK, 2-SHELF           125.00   1
     13 87-114   C-500-6000 LAMP, FLOOR           BRASS, 6-FOOT, ART DECO       150.00   3
     14 87-115   C-500-6050 LAMP, FLOOR           BRASS, 6-FOOT, ENGLISH        165.00   1
     15 87-116   C-600-5000 DESK,EXECUTIVE 5-FOOT WOOD, OAK, FANCY             1500.00   1
     16 87-116   C-700-2030 FILE CABINET,2 DRAWER METAL, BLACK                   75.00   1
     17 87-116   C-222-1001 CHAIR, DESK           LEATHER, BROWN               1000.00   1
```

The following is a list of the index keys used for the Stock.dbf file:

```
Production MDX file: STOCK.MDX
          Index TAG: ORDER_ID  Key: ORDER_ID
          Index TAG: PART_NAME  Key: PART_NAME
```

## Menus.prg

The following is a listing of Menus.prg. Figure C-1 shows how the screen looks after you run this program.

```
* Menus.prg
SET TALK OFF
CLEAR
Medit = .F.
DO Def_mens
ON PAD View OF Main ACTIVATE POPUP View_pop
ON PAD Goto OF Main ACTIVATE POPUP Goto_pop
ON PAD Print OF Main ACTIVATE POPUP Prin_pop
ON SELECTION PAD Exit OF Main ACTIVATE POPUP Exit_pop
ON SELECTION POPUP Exit_pop DO Exit_pro
ON SELECTION POPUP View_pop DO View_pro
ACTIVATE MENU Main PAD View
CLEAR ALL
SET TALK ON
RETURN

PROCEDURE Def_mens
*—— Menu Main
DEFINE MENU Main
DEFINE PAD View OF Main PROMPT "Add/Edit" AT 2,4
DEFINE PAD Goto OF Main PROMPT "Goto/Search" AT 2,16
DEFINE PAD Print OF Main PROMPT "Print" AT 2,30
DEFINE PAD Exit OF Main PROMPT "Exit" AT 2,38
*—— Popup View_pop
DEFINE POPUP View_pop FROM 3,4 TO 8,19
DEFINE BAR 1 OF View_pop PROMPT "Add new record"
DEFINE BAR 2 OF View_pop PROMPT "Edit"
DEFINE BAR 3 OF View_pop PROMPT REPLICATE("-",16) SKIP
DEFINE BAR 4 OF View_pop PROMPT "Delete" SKIP FOR Medit
*—— Popup Goto_pop
DEFINE POPUP Goto_pop FROM 3,16 TO 6,28
DEFINE BAR 1 OF Goto_pop PROMPT "Skip"
DEFINE BAR 2 OF Goto_pop PROMPT "Jump to"
*—— Popup Prin_pop
DEFINE POPUP Prin_pop FROM 3,30 TO 7,42
DEFINE BAR 1 OF Prin_pop PROMPT "Destination"
DEFINE BAR 2 OF Prin_pop PROMPT "Options"
DEFINE BAR 3 OF Prin_pop PROMPT "Eject page"
*—— Popup Exit_pop
DEFINE POPUP Exit_pop FROM 3,38 TO 6,57
DEFINE BAR 1 OF Exit_pop PROMPT "Quit"
DEFINE BAR 2 OF Exit_pop PROMPT "Exit to dot prompt"
RETURN
```

```
PROCEDURE Exit_pro
DO CASE
   CASE BAR() = 1
        QUIT
        CANCEL
   CASE BAR() = 2
        DEACTIVATE MENU
ENDCASE
RETURN

PROCEDURE View_pro
DO CASE
   CASE BAR() = 1
        APPEND BLANK
        EDIT NEXT 1
   CASE BAR() = 2
        EDIT NEXT 1
        Medit = .NOT. Medit
   CASE BAR() = 4
        DELETE
ENDCASE
RETURN
* EOP: Menus.prg
```

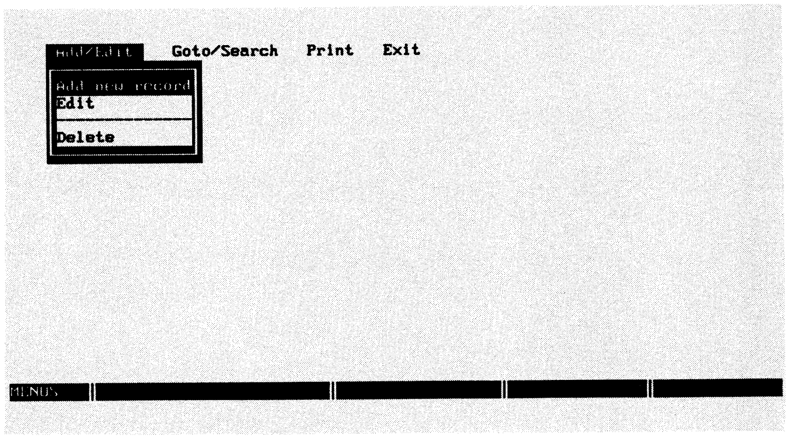


Figure C-1  Output of Menus.prg

# File Extensions

The first part of this appendix lists all the dBASE IV file extensions and briefly describes their use. The second part shows the relationship between uncompiled and compiled files, and notes backward compatibility with dBASE III PLUS.

## File Extensions Used by dBASE IV

dBASE IV uses a two- or three- letter file extension following the period in a filename. Any valid DOS filename can be used. (See Chapter 1, "Essentials," for a complete description of filenames.) Table D-1 lists the extensions used and provides a brief description of the file content.

Table D-1 dBASE IV file extensions

| Extension | File Content |
|---|---|
| .$$$ | Temporary file; action was not complete |
| .app | Application design object file; Applications Generator only |
| .bak | Command, procedure, or database backup file |
| .bar | Horizontal bar design object file; Applications Generator only |
| .bch | Batch process design object file; Applications Generator only |
| .bin | Binary file |
| .cat | Catalog file |
| .cht | CHART-MASTER® file; used with dBASE/CHART-MASTER Bridge |
| .cod | Template source file |
| .cpt | Encrypted memo file; used with password information (.crp) file |
| .crp | Password information file; created with PROTECT only |
| .cvt | Convert file; for multi-user change detection file |
| .db | Configuration file; for defaults on dBASE IV start-up |
| .db2 | Renamed old dBASE II file; used for import and export |

(continued)

Table D-1 dBASE IV file extensions (*continued*)

| **Extension** | **File Content** |
|---|---|
| .dbf | Database file |
| .dbo | Command and procedure object file |
| .dbt | Database memo file |
| .def | Selector definition file |
| .dif | Data Interchange Format, or VisiCalc file; used with APPEND FROM/COPY TO |
| .doc | Documentation file; Applications Generator only |
| .fil | Files list design object file |
| .fmo | Compiled format (.fmt) file |
| .fmt | Generated format file; from .scr file |
| .fr3 | Renamed old dBASE III report form (.frm) file |
| .frg | Generated report form file; from .frm file |
| .frm | Report form file |
| .fro | Compiled report form file (.frg) file |
| .fw2 | Framework spreadsheet/database file; used for import/export |
| .gen | Template file |
| .key | Keystroke macro library file |
| .lb3 | Renamed old dBASE III label form (.lbl) file |
| .lbg | Generated label form file; from .lbl file |
| .lbl | Label form file |
| .lbo | Compiled label form (.lbl) file |
| .log | Transaction log file |
| .mdx | Multiple index file |
| .mem | Memory file |
| .ndx | Single index file |
| .npi | Reports, forms, label files; template interpreter only |
| .pop | Pop-up menu design object file; Applications Generator only |
| .pr2 | Printer driver file |
| .prf | Print form file |
| .prg | dBASE command or procedure file |
| .prs | dBASE/SQL command or procedure file |

(*continued*)

Table D-1 dBASE IV file extensions (*continued*)

| **Extension** | **File Content** |
|---|---|
| .prt | Printer output file |
| .qbe | QBE query file |
| .qbo | Compiled QBE query (.qbe) file |
| .qry | Query file |
| .rpd | RapidFile file; used for import/export |
| .sc3 | Renamed old dBASE III screen (.scr) file |
| .scr | Screen file |
| .str | Structure list design object file; Applications Generator only |
| .t44/.w44 | Intermediate work files; used by SORT and INDEX |
| .tbk | Database memo backup file |
| .txt | ASCII text output file |
| .upd | QBE update query file |
| .upo | Compiled QBE update query (.upd) file |
| .val | Values list design object file; Applications Generator only |
| .vue | View file |
| .win | Logical window save file |
| .wks | Lotus 1-2-3 file; used with APPEND FROM/COPY TO |

## File Relations and Compatibilities

Table D-2 shows the relationships among various dBASE files and their associated generated and compiled files. The table also shows whether or not the dBASE IV file is compatible with dBASE III PLUS.

Table D-2 File relations and compatibilities

| **File Type** | **dBASE IV Extension** | **Compiled Object File** | **Backward Compatibility** | **Rename from dBASE III PLUS** |
|---|---|---|---|---|
| Catalog | .cat | | yes | |
| Database file | .dbf | | yes * | |
| Memo | .dbt | | yes | |
| Form, design | .scr | | no | .sc3 |
| Form, code generation | .fmt | .fmo | no ** | |
| Report, design | .frm | | no | .fr3 |
| Report, code generation | .frg | .fro | no | |
| Label, design | .lbl | | no | .lb3 |
| Label, code generation | .lbg | .lbo | no | |
| Queries | .qbe | .qbo | no | |
| Update query | .upd | .dbo | no | |
| SQL | .prs | .dbo | no | |
| Templates | .gen | | no | |
| Macros | .key | | no | |

* dBASE IV database files that contain type F numbers are not compatible with dBASE III PLUS. Also, numeric accuracy may be lost if a .dbf file is used in dBASE III PLUS, because dBASE IV allows numeric fields to contain up to 20 digits. (dBASE III PLUS allowed only 19 digits in a numeric field.)

** dBASE III PLUS may use your dBASE IV format file if it contains only comment lines or @ commands, and if the @ commands do not contain new dBASE IV keywords (such as VALID or WHEN).

# Structure of a Database (.dbf) File

## Database Header and Records

A database (.dbf) file is composed of a header, data records, deletion flags, and an end-of-file marker. The header contains information about the file structure, and the records contain the actual data. One byte of each record is reserved for the deletion flag.

## Database Header Structure

The header structure, detailed in Tables E-1 and E-2, provides information dBASE IV uses to maintain the database file.

Table E-1 Database file header

| Byte | Contents | Meaning |
|---|---|---|
| 0 | 1 byte | Valid dBASE IV file; bits 0–2 indicate version number, bits 3–5 are reserved for SQL, bits 6 and 7 indicate the presence of a memo file |
| 1–3 | 3 bytes | Date of last update; formatted as YYMMDD |
| 4–7 | 32-bit number | Number of records in the database file |
| 8–9 | 16-bit number | Number of bytes in the header |
| 10–11 | 16-bit number | Number of bytes in the record |
| 12–13 | 2 bytes | Reserved |
| 14 | 1 byte | Flag indicating incomplete transaction* |
| 15 | 1 byte | Encryption flag** |
| 16–27 | 12 bytes | Reserved for dBASE IV on a local area network |
| 28 | 1 byte | Production .mdx file flag; 01H if there is a production .mdx file, 00H if not |
| 29–31 | 3 bytes | Reserved |
| 32–n*** | 32 bytes each | Field descriptor array (the structure of this array is shown in Table E-2) |
| n + 1 | 1 byte | 0DH as the field terminator |

* The ISMARKED() function checks this flag. BEGIN TRANSACTION sets it to 01, END TRANSACTION and ROLLBACK reset it to 00.

** If this flag is set to 01H, the message **Database encrypted** appears. Changing this flag to 00H removes the message, but does not decrypt the file.

*** n is the last byte in the field descriptor array. The size of the array depends on the number of fields in the database file.

Table E-2 Database field descriptor bytes

| Byte | Contents | Meaning |
|---|---|---|
| 0–10 | 11 bytes | Field name in ASCII (zero-filled) |
| 11 | 1 byte | Field type in ASCII (C, D, F, L, M, or N) |
| 12–15 | 4 bytes | Reserved |
| 16 | 1 byte | Field length in binary |
| 17 | 1 byte | Field decimal count in binary |
| 18–19 | 2 bytes | Reserved |
| 20 | 1 byte | Work area ID |
| 21–31 | 11 bytes | Reserved |

## Database Records

The records follow the header in the database file. Data records are preceded by one byte; that is, a space (20H) if the record is not deleted, an asterisk (2AH) if the record is deleted. Fields are packed into records without field separators or record terminators. The end of the file is marked by a single byte, with the end-of-file marker an ASCII 26 (1AH) character.

You can input ASCII data as indicated in Table E-3.

Table E-3 Allowable input for each data type

| Data Type | What it Accepts |
|---|---|
| C (Character) | All ASCII characters |
| D (Date) | Numbers and a character to separate month, day, and year (stored internally as 8 digits in YYYYMMDD format) |
| F (Floating point binary numeric) | – . 0 1 2 3 4 5 6 7 8 9 |
| L (Logical) | ? Y y N n T t F f (? when not initialized) |
| M (Memo) | All ASCII characters (stored internally as 10 digits representing a .dbt block number) |
| N (Binary coded decimal numeric) | – . 0 1 2 3 4 5 6 7 8 9 |

## Memo Fields and the .dbt File

A memo (.dbt) file consists of blocks numbered sequentially (0, 1, 2, and so on). SET BLOCKSIZE determines the size of each block. The first block in the memo file, block 0, is the memo file header.

Each memo field of each record in the .dbf file contains the number of the block (in ASCII) where the memo field begins. If the memo field contains no data, the .dbf file contains blanks (20H) rather than a number.

When data is changed in a memo field, the block numbers may also change, and the number in the .dbf may be changed to reflect the new location.

Unlike dBASE III PLUS, if you delete text in a memo field, dBASE IV may reuse the space from the deleted text when you input new text. dBASE III PLUS always appended new text to the end of the .dbt file. In dBASE III PLUS, the .dbt file size grew whenever new text was added, even if other text in the file was deleted.

# dBASE IV Printer Drivers

You can configure dBASE IV to work with most available printers by installing the printer driver files supplied on the installation disk. The installation program prompts you for the printer model you will be using, and installs the corresponding printer driver file. You may also change the default printer drivers in the Config.db file, or use _pdriver = <printer driver filename> from the dot prompt, at any time after installation.

dBASE IV supports up to four different printer drivers, with five fonts each, in Config.db. Refer to Chapter 6, "Customizing dBASE IV," for information on the Config.db file, and Chapter 5, "System Memory Variables," for information on the _pdriver system memory variable.

The following table shows the printer driver files available on your installation disk, and provides information about the type styles each driver supports.

Table F-1 dBASE IV printer drivers

| Manufacturer | Model | Driver Filename |
|---|---|---|
| AMT | AMT 250 | AMT.pr2 |
| Anadex | 6000 | ANA6000.pr2 |
| | 9625B (Silent Scribe) | ANA9625B.pr2 |
| ASCII * | | ASCII.pr2 |
| Brother | HR-15, HR-25 | HR15.pr2 |
| Canon USA, Inc. | A2 in 150 dpi graphics mode | CANA2_15.pr2 |
| C.Itoh Digital Products, Inc. | 1550A, B, BPI | CI1550A.pr2 |
| | 1550S, SC | CI1550S.pr2 |
| | 715 | CI715_2.pr2 |
| | 815 (P351 Mode) | CI815.pr2 |
| | 8510A, B, BPI | CI8510A.pr2 |
| | 8510S, SC | CI8510S.pr2 |
| | 3500 model 20 | CI3520.pr2 |
| | ProWriter Jr. | FX80_1.pr2 |
| Datasouth | 180 | DS180.pr2 |
| | 220 | DS220.prR |
| DataProducts | P-80, P-132 | DP_P80.pr2 |
| | SPG-8050, SPG-8070 | DP_8050.pr2 |
| Diablo | (Xerox) 630 API | DIAB630A.pr2 |
| Epson America, Inc. | FX-85, FX-185 Lo-res graphics | FX85_1.pr2 |
| | FX-86E Lo-res Graphics | FX86E_1.pr2 |
| | GQ-3500 Standard-resolution | GQ35_150.pr2 |
| | JX-80 | JX80_2.pr2 |
| | JX-80 with 8177 | EPSN8177.pr2 |
| | LX-80, LX-90 | LX80.pr2 |
| | MX-80, MX-100 | Generic.pr2 |
| | MX-80/MX-100 with Graftrax PLUS | MX80G.pr2 |
| | RX-80, FX-80, FX-80 PLUS | FX80_1.pr2 |
| | RX-80/FX-80 with 8177 | EPSN8177.pr2 |
| | RX-100, FX-100, FX-100 PLUS | FX80_1.pr2 |
| | RX-100/FX-100 with 8177 | EPSN8177.pr2 |
| | LQ-800 | LQ800_2.pr2 |
| | LQ-1500, SQ-1500 | LQ1500_1.pr2 |
| | DX-10, DX-20, DX-35 | DIAB630A.pr2 |
| Facit | 4512 | FAC4512.pr2 |
| Fujitsu | DotMax 9f Lo-res graphics | FX80_1.pr2 |
| | DotMax 9i | IBMGP.pr2 |
| | DotMax 24i | FUJ24I.pr2 |
| | DotMAx 24c | FUJ24C.pr2 |

* The ASCII driver allows you to generate ASCII files that do not contain printer escape codes.

| Bold | Italic | Underline | Elite | Compressed | Letter Quality |
|---|---|---|---|---|---|
| Double | Yes | Dash | No | No | Yes |
| Double | Yes | Yes | Yes | Yes | Yes |
| Yes | Dash | Dash | Yes | Yes | Yes |
| Yes | Underline | Yes | Yes | Yes | N/A |
| Yes | Yes | Yes | Yes | Yes | Yes |
| Yes | Underline | Yes | Yes | Yes | No |
| Yes | Yes | Yes | Yes | Yes | No |
| Yes | Yes | Yes | Yes | Yes | Yes |
| Yes | Yes | Yes | Yes | Yes | Yes |
| Yes | Underline | Yes | Yes | Yes | No |
| Yes | Yes | Yes | Yes | Yes | No |
| Yes | Yes | Yes | Yes | Yes | Yes |
| Yes | Yes | Yes | Yes | Yes | Yes |
| Double | Underline | Dash | Yes | Yes | No |
| Double | Dash | Dash | No | Yes | Yes |
| Double | Dash | Dash | Yes | Yes | Yes |
| Yes | Underline | Yes | Yes | Yes | Yes |
| Yes | Underline | Yes | No | No | N/A |
| Yes | Yes | Yes | Yes | Yes | Yes |
| Yes | Yes | Yes | Yes | Yes | Yes |
| Yes | Underline | Yes | Yes | Yes | N/A |
| Yes | Yes | Yes | Yes | Yes | Yes |
| Double | Yes | Yes | Yes | Yes | Yes |
| Yes | Yes | Yes | Yes | Yes | Yes |
| Double | Dash | Dash | No | No | No |
| Yes | No | Yes | No | Yes | Yes |
| Yes | Yes | Yes | Yes | Yes | Yes |
| Double | Yes | Yes | Yes | Yes | Yes |
| Yes | Yes | Yes | Yes | Yes | Yes |
| Double | Yes | Yes | Yes | Yes | Yes |
| Yes | Yes | Yes | Yes | Yes | Yes |
| Yes | Yes | Yes | Yes | Yes | Yes |
| Yes | Underline | Yes | No | No | N/A |
| Yes | Underline | Yes | No | Yes | Yes |
| Yes | Yes | Yes | Yes | Yes | Yes |
| Yes | Underline | Yes | No | Yes | Yes |
| Yes | Yes | Yes | Yes | Yes | Yes |
| Yes | Yes | Yes | Yes | Yes | Yes |

*(continued)*

Table F-1 dBASE IV printer drivers (*continued*)

| **Manufacturer** | **Model** | **Driver Filename** |
|---|---|---|
| Generic Driver ** | Any printer not listed | GENERIC.pr2 |
| Hewlett-Packard, Inc. | DeskJet 150-dpi | HPDSK150.pr2 |
| | ThinkJet | HPJET.pr2 |
| | LaserJet | |
| | Portrait Orientation | HPLAS100.pr2 |
| | Landscape | HPLASL.pr2 |
| | QuietJet standard-res | HPQJET1.pr2 |
| International Business Machines Corporation | Color JetPrinter | IBMCJET2.pr2 |
| | Graphics Printer | IBMGP.pr2 |
| | QuietWriter (model 1) | IBMQUIET.pr2 |
| | QuietWriter (model 2) | IBMQ2_1.pr2 |
| | QuietWriter (model 3) | IBMQ3_1.pr2 |
| | ProPrinter | IBMPRO_1.pr2 |
| | WheelPrinter | IBMWHEEL.pr2 |
| Nippon Electric Corporation | Pinwriter P2, P3 | IBMGP.pr2 |
| | Pinwriter P5 | NECP5.pr2 |
| | Spinwriter 3550 | NEC3550.pr2 |
| | Spinwriter 7710 | NEC7710.pr2 |
| | Spinwriter 8850 | NEC8850.pr2 |
| Oki America, Inc. | Microline 80, 82, 83A | GENERIC.pr2 |
| | Microline 82A/83A with Okigraph | OKI82AGR.pr2 |
| | Microline 84 | OKI84.pr2 |
| | Microline 182, 183 | OKI182.pr2 |
| | 182/183 with Plug 'n Play | OKI182PN.pr2 |
| | Microline 92, 93 | OKI92_2.pr2 |
| | Microline 192, 193 | OKI192_2.pr2 |
| | 92/93/192/193 with Plug 'n Play | OKI92PNP.pr2 |
| | Okimate-20 | IBMGP.pr2 |
| | Pacemark 2350, 2410 | OKI2410.pr2 |
| Qume | Sprint 11 Plus, LetterPro 20 | QUME11.pr2 |
| Ricoh | 6000 Portrait Orient (100 dpi) | RIC6K10.pr2 |
| | 6000 Landscape Orient (100 dpi) | RIC6K10L.pr2 |
| Star Micronics | Gemini-10x, Gemini-15x | GEM10.pr2 |
| Tandy Corporation | CGP-220 | CGP220.pr2 |
| Texas Instruments, Inc. | 855, 865 | TI855.pr2 |
| Toshiba America, Inc. | P341, P351 | P351.pr2 |
| | P1340, P1351 | P1340.pr2 |
| | PageLaser/2 (150 dpi) | TOSLAS15.pr2 |
| Xerox Corporation | (Diablo) 630 API | DIAB630A.pr2 |

** The Generic driver provides basic styling options across a wide range of printer models.

| Bold | Italic | Underline | Elite | Compressed | Letter Quality |
|---|---|---|---|---|---|
| Double | Dash | Dash | No | No | No |
| Yes | Yes | Yes | Yes | Yes | Yes |
| Yes | Underline | Yes | No | Yes | No |
| Double | Underline | Yes | Yes | No | N/A |
| Double | Underline | Yes | Yes | Yes | N/A |
| Yes | Underline | Yes | Yes | Yes | No |
| Yes | Underline | Yes | No | Yes | Yes |
| Yes | Underline | Yes | No | Yes | Yes |
| Double | Underline | Yes | No | No | No |
| Double | Underline | Yes | No | No | No |
| Yes | Underline | Yes | Yes | Yes | Yes |
| Yes | Underline | Yes | Yes | Yes | Yes |
| Double | Underline | Yes | Yes | Yes | N/A |
| Yes | Underline | Yes | No | Yes | Yes |
| Yes | Yes | Dash | Yes | Yes | Yes |
| Double | Underline | Yes | Yes | Yes | N/A |
| Yes | Underline | Yes | No | No | N/A |
| Yes | Underline | Yes | Yes | Yes | N/A |
| Double | Underline | Yes | No | No | No |
| Double | Dash | Dash | No | Yes | No |
| Double | Underline | Yes | Yes | Yes | Yes |
| Double | Underline | Yes | Yes | Yes | Yes |
| Yes | Underline | Yes | No | Yes | Yes |
| Yes | Underline | Yes | Yes | Yes | Yes |
| Yes | Yes | Yes | Yes | Yes | Yes |
| Yes | Underline | Yes | No | Yes | Yes |
| Yes | Underline | Yes | No | Yes | Yes |
| Double | Underline | Yes | Yes | Yes | Yes |
| Yes | Underline | Yes | Yes | Yes | N/A |
| Yes | Underline | Yes | No | Yes | N/A |
| Yes | Underline | Yes | No | Yes | N/A |
| Yes | Yes | Yes | Yes | Yes | Yes |
| No | No | No | Yes | No | No |
| Yes | Yes | Dash | Yes | Yes | Yes |
| Yes | Yes | Yes | Yes | Yes | Yes |
| Double | Underline | Yes | Yes | Yes | Yes |
| Yes | Yes | Yes | No | Yes | N/A |
| Yes | Underline | Yes | No | No | No |

# ASCII Chart

| Binary | Hex | Decimal | Character | Code | Symbol | Description |
|---|---|---|---|---|---|---|
| 00000000 | 00 | 0 | | ^@ | NUL | Null |
| 00000001 | 01 | 1 | ☺ | ^A | SOH | Start of Heading |
| 00000010 | 02 | 2 | ☻ | ^B | STX | Start of Text |
| 00000011 | 03 | 3 | ♥ | ^C | ETX | End of Text |
| 00000100 | 04 | 4 | ♦ | ^D | EOT | End of Transmission |
| 00000101 | 05 | 5 | ♣ | ^E | ENQ | Enquiry |
| 00000110 | 06 | 6 | ♠ | ^F | ACK | Acknowledge |
| 00000111 | 07 | 7 | • | ^G | BEL | Bell |
| 00001000 | 08 | 8 | ◘ | ^H | BS | Backspace |
| 00001001 | 09 | 9 | ○ | ^I | SH | Horizontal Tabulation |
| 00001010 | 0A | 10 | ◙ | ^J | LF | Line Feed |
| 00001011 | 0B | 11 | ♂ | ^K | VT | Vertical Tabulation |
| 00001100 | 0C | 12 | ♀ | ^L | FF | Form Feed |
| 00001101 | 0D | 13 | ♪ | ^M | CR | Carriage Return |
| 00001110 | 0E | 14 | ♫ | ^N | SO | Shift Out |
| 00001111 | 0F | 15 | ☼ | ^O | SI | Shift In |
| 00010000 | 10 | 16 | ► | ^P | DLE | Data Link Escape |
| 00010001 | 11 | 17 | ◄ | ^Q | DC1 | Device Control 1 |
| 00010010 | 12 | 18 | ↕ | ^R | DC2 | Device Control 2 |
| 00010011 | 13 | 19 | ‼ | ^S | DC3 | Device Control 3 |
| 00010100 | 14 | 20 | ¶ | ^T | DC4 | Device Control 4 |
| 00010101 | 15 | 21 | § | ^U | NAK | Negative Acknowledge |
| 00010110 | 16 | 22 | ▬ | ^V | SYN | Synchronous Idle |
| 00010111 | 17 | 23 | ↨ | ^W | ETB | End of Transmission Block |
| 00011000 | 18 | 24 | ↑ | ^X | CAN | Cancel |
| 00011001 | 19 | 25 | ↓ | ^Y | EM | End of Medium |

| Binary | Hex | Decimal | Character | Code | Symbol | Description |
|---|---|---|---|---|---|---|
| 00011010 | 1A | 26 | → | ^Z | SUB | Substitute |
| 00011011 | 1B | 27 | ← | ^[ | ESC | Escape * |
| 00011100 | 1C | 28 | ∟ | ^\ | FS | File Separator |
| 00011101 | 1D | 29 | ↔ | ^] | GS | Group Separator |
| 00011110 | 1E | 30 | ▲ | ^^ | RS | Record Separator |
| 00011111 | 1F | 31 | ▼ | ^- | US | Unit Separator |
| 00100000 | 20 | 32 | | | | |
| 00100001 | 21 | 33 | ! | | | |
| 00100010 | 22 | 34 | " | | | |
| 00100011 | 23 | 35 | # | | | |
| 00100100 | 24 | 36 | $ | | | |
| 00100101 | 25 | 37 | % | | | |
| 00100110 | 26 | 38 | & | | | |
| 00100111 | 27 | 39 | ' | | | |

* Escape cannot be trapped

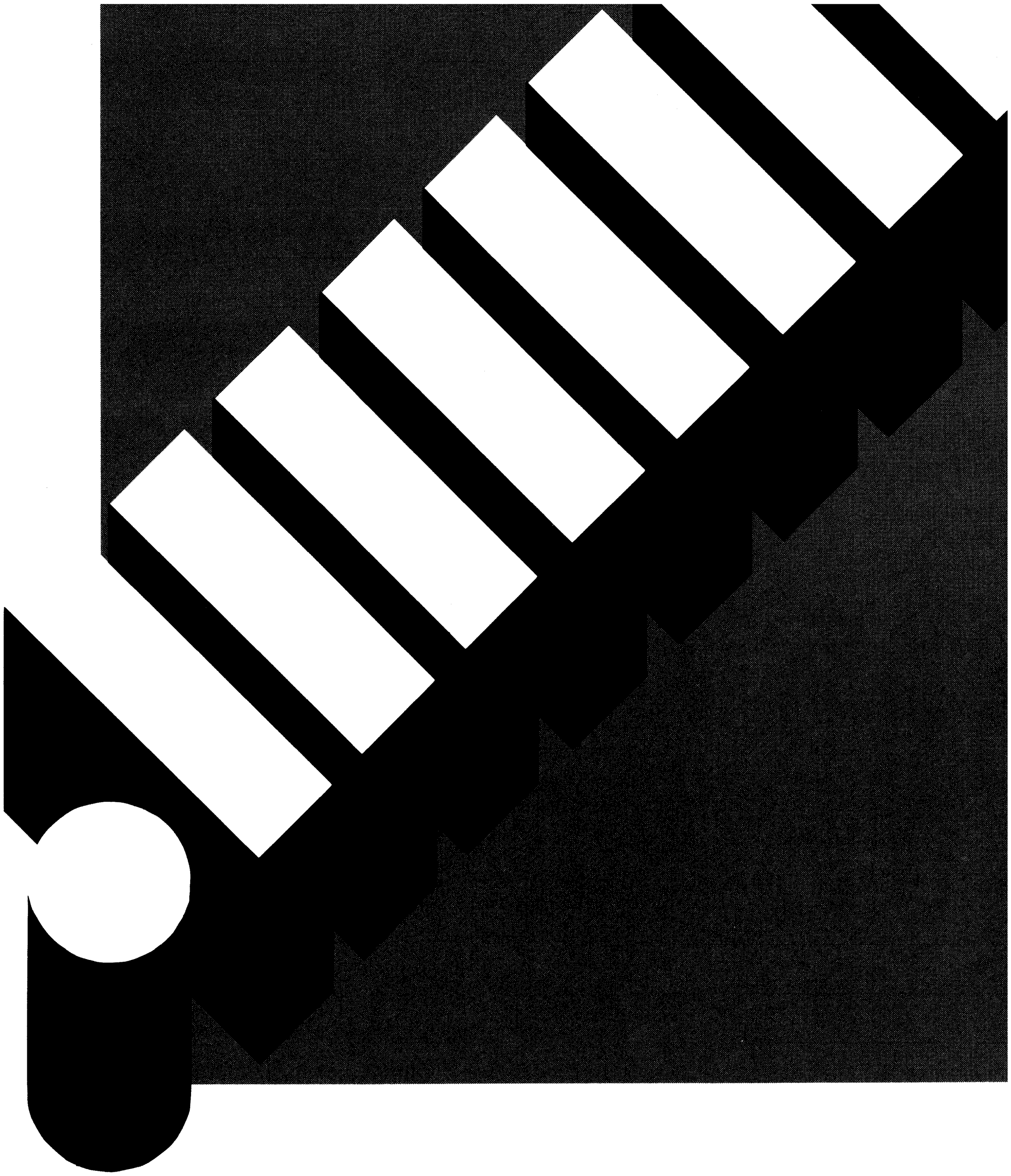
| Binary | Hex | Decimal | Character | Binary | Hex | Decimal | Character |
|---|---|---|---|---|---|---|---|
| 00101000 | 28 | 40 | ( | 00110111 | 37 | 55 | 7 |
| 00101001 | 29 | 41 | ) | 00111000 | 38 | 56 | 8 |
| 00101010 | 2A | 42 | * | 00111001 | 39 | 57 | 9 |
| 00101011 | 2B | 43 | + | 00111010 | 3A | 58 | : |
| 00101100 | 2C | 44 | , | 00111011 | 3B | 59 | ; |
| 00101101 | 2D | 45 | - | 00111100 | 3C | 60 | < |
| 00101110 | 2E | 46 | . | 00111101 | 3D | 61 | = |
| 00101111 | 2F | 47 | / | 00111110 | 3E | 62 | > |
| 00110000 | 30 | 48 | 0 | 00111111 | 3F | 63 | ? |
| 00110001 | 31 | 49 | 1 | 01000000 | 40 | 64 | @ |
| 00110010 | 32 | 50 | 2 | 01000001 | 41 | 65 | A |
| 00110011 | 33 | 51 | 3 | 01000010 | 42 | 66 | B |
| 00110100 | 34 | 52 | 4 | 01000011 | 43 | 67 | C |
| 00110101 | 35 | 53 | 5 | 01000100 | 44 | 68 | D |
| 00110110 | 36 | 54 | 6 | 01000101 | 45 | 69 | E |

| Binary | Hex | Decimal | Character |
|---|---|---|---|
| 01000110 | 46 | 70 | F |
| 01000111 | 47 | 71 | G |
| 01001000 | 48 | 72 | H |
| 01001001 | 49 | 73 | I |
| 01001010 | 4A | 74 | J |
| 01001011 | 4B | 75 | K |
| 01001100 | 4C | 76 | L |
| 01001101 | 4D | 77 | M |
| 01001110 | 4E | 78 | N |
| 01001111 | 4F | 79 | O |
| 01010000 | 50 | 80 | P |
| 01010001 | 51 | 81 | Q |
| 01010010 | 52 | 82 | R |
| 01010011 | 53 | 83 | S |
| 01010100 | 54 | 84 | T |
| 01010101 | 55 | 85 | U |
| 01010110 | 56 | 86 | V |
| 01010111 | 57 | 87 | W |
| 01011000 | 58 | 88 | X |
| 01011001 | 59 | 89 | Y |
| 01011010 | 5A | 90 | Z |
| 01011011 | 5B | 91 | [ |
| 01011100 | 5C | 92 | \ |
| 01011101 | 5D | 93 | ] |
| 01011110 | 5E | 94 | ^ |
| 01011111 | 5F | 95 | _ |
| 01100000 | 60 | 96 | \′ |
| 01100001 | 61 | 97 | a |
| 01100010 | 62 | 98 | b |
| 01100011 | 63 | 99 | c |
| 01100100 | 64 | 100 | d |
| 01100101 | 65 | 101 | e |
| 01100110 | 66 | 102 | f |
| 01100111 | 67 | 103 | g |
| 01101000 | 68 | 104 | h |
| 01101001 | 69 | 105 | i |
| 01101010 | 6A | 106 | j |
| 01101011 | 6B | 107 | k |
| 01101100 | 6C | 108 | l |
| 01101101 | 6D | 109 | m |
| 01101110 | 6E | 110 | n |
| 01101111 | 6F | 111 | o |
| 01110000 | 70 | 112 | p |
| 01110001 | 71 | 113 | q |
| 01110010 | 72 | 114 | r |
| 01110011 | 73 | 115 | s |
| 01110100 | 74 | 116 | t |
| 01110101 | 75 | 117 | u |
| 01110110 | 76 | 118 | v |
| 01110111 | 77 | 119 | w |
| 01111000 | 78 | 120 | x |
| 01111001 | 79 | 121 | y |
| 01111010 | 7A | 122 | z |
| 01111011 | 7B | 123 | { |
| 01111100 | 7C | 124 | \| |
| 01111101 | 7D | 125 | } |
| 01111110 | 7E | 126 | ~ |
| 01111111 | 7F | 127 | ⌂ |
| 10000000 | 80 | 128 | Ç |
| 10000001 | 81 | 129 | ü |
| 10000010 | 82 | 130 | é |
| 10000011 | 83 | 131 | â |
| 10000100 | 84 | 132 | ä |
| 10000101 | 85 | 133 | à |

| Binary | Hex | Decimal | Character |
|---|---|---|---|
| 10000110 | 86 | 134 | å |
| 10000111 | 87 | 135 | ç |
| 10001000 | 88 | 136 | ê |
| 10001001 | 89 | 137 | ë |
| 10001010 | 8A | 138 | è |
| 10001011 | 8B | 139 | ï |
| 10001100 | 8C | 140 | î |
| 10001101 | 8D | 141 | ì |
| 10001110 | 8E | 142 | Ä |
| 10001111 | 8F | 143 | Å |
| 10010000 | 90 | 144 | É |
| 10010001 | 91 | 145 | æ |
| 10010010 | 92 | 146 | Æ |
| 10010011 | 93 | 147 | ô |
| 10010100 | 94 | 148 | ö |
| 10010101 | 95 | 149 | ò |
| 10010110 | 96 | 150 | û |
| 10010111 | 97 | 151 | ù |
| 10011000 | 98 | 152 | ÿ |
| 10011001 | 99 | 153 | Ö |
| 10011010 | 9A | 154 | Ü |
| 10011011 | 9B | 155 | ¢ |
| 10011100 | 9C | 156 | £ |
| 10011101 | 9D | 157 | ¥ |
| 10011110 | 9E | 158 | ₧ |
| 10011111 | 9F | 159 | ƒ |
| 10100000 | A0 | 160 | á |
| 10100001 | A1 | 161 | í |
| 10100010 | A2 | 162 | ó |
| 10100011 | A3 | 163 | ú |
| 10100100 | A4 | 164 | ñ |
| 10100101 | A5 | 165 | Ñ |
| 10100110 | A6 | 166 | ª |
| 10100111 | A7 | 167 | º |
| 10101000 | A8 | 168 | ¿ |
| 10101001 | A9 | 169 | ⌐ |
| 10101010 | AA | 170 | ¬ |
| 10101011 | AB | 171 | ½ |
| 10101100 | AC | 172 | ¼ |
| 10101101 | AD | 173 | ¡ |
| 10101110 | AE | 174 | « |
| 10101111 | AF | 175 | » |
| 10110000 | B0 | 176 | ░ |
| 10110001 | B1 | 177 | ▒ |
| 10110010 | B2 | 178 | ▓ |
| 10110011 | B3 | 179 | │ |
| 10110100 | B4 | 180 | ┤ |
| 10110101 | B5 | 181 | ╡ |
| 10110110 | B6 | 182 | ╢ |
| 10110111 | B7 | 183 | ╖ |
| 10111000 | B8 | 184 | ╕ |
| 10111001 | B9 | 185 | ╣ |
| 10111010 | BA | 186 | ║ |
| 10111011 | BB | 187 | ╗ |
| 10111100 | BC | 188 | ╝ |
| 10111101 | BD | 189 | ╜ |
| 10111110 | BE | 190 | ╛ |
| 10111111 | BF | 191 | ┐ |
| 11000000 | C0 | 192 | └ |
| 11000001 | C1 | 193 | ┴ |
| 11000010 | C2 | 194 | ┬ |
| 11000011 | C3 | 195 | ├ |
| 11000100 | C4 | 196 | ─ |
| 11000101 | C5 | 197 | ┼ |

| Binary | Hex | Decimal | Character |
|---|---|---|---|
| 11000110 | C6 | 198 | ╞ |
| 11000111 | C7 | 199 | ╟ |
| 11001000 | C8 | 200 | ╚ |
| 11001001 | C9 | 201 | ╔ |
| 11001010 | CA | 202 | ╩ |
| 11001011 | CB | 203 | ╦ |
| 11001100 | CC | 204 | ╠ |
| 11001101 | CD | 205 | ═ |
| 11001110 | CE | 206 | ╬ |
| 11001111 | CF | 207 | ╧ |
| 11010000 | D0 | 208 | ╨ |
| 11010001 | D1 | 209 | ╤ |
| 11010010 | D2 | 210 | ╥ |
| 11010011 | D3 | 211 | ╙ |
| 11010100 | D4 | 212 | ╘ |
| 11010101 | D5 | 213 | ╒ |
| 11010110 | D6 | 214 | ╓ |
| 11010111 | D7 | 215 | ╫ |
| 11011000 | D8 | 216 | ╪ |
| 11011001 | D9 | 217 | ┘ |
| 11011010 | DA | 218 | ┌ |
| 11011011 | DB | 219 | █ |
| 11011100 | DC | 220 | ▄ |
| 11011101 | DD | 221 | ▌ |
| 11011110 | DE | 222 | ▐ |
| 11011111 | DF | 223 | ▀ |
| 11100000 | E0 | 224 | α |
| 11100001 | E1 | 225 | β |
| 11100010 | E2 | 226 | Γ |
| 11100011 | E3 | 227 | π |
| 11100100 | E4 | 228 | Σ |
| 11100101 | E5 | 229 | σ |
| 11100110 | E6 | 230 | µ |
| 11100111 | E7 | 231 | τ |
| 11101000 | E8 | 232 | Φ |
| 11101001 | E9 | 233 | Θ |
| 11101010 | EA | 234 | Ω |
| 11101011 | EB | 235 | δ |
| 11101100 | EC | 236 | ∞ |
| 11101101 | ED | 237 | φ |
| 11101110 | EE | 238 | ε |
| 11101111 | EF | 239 | ∩ |
| 11110000 | F0 | 240 | ≡ |
| 11110001 | F1 | 241 | ± |
| 11110010 | F2 | 242 | ≥ |
| 11110011 | F3 | 243 | ≤ |
| 11110100 | F4 | 244 | ⌠ |
| 11110101 | F5 | 245 | ⌡ |
| 11110110 | F6 | 246 | ÷ |
| 11110111 | F7 | 247 | ≈ |
| 11111000 | F8 | 248 | ° |
| 11111001 | F9 | 249 | ∙ |
| 11111010 | FA | 250 | · |
| 11111011 | FB | 251 | √ |
| 11111100 | FC | 252 | ⁿ |
| 11111101 | FD | 253 | ² |
| 11111110 | FE | 254 | ■ |
| 11111111 | FF | 255 | |

# Index



i 1 2 3 4 5 6 A B C

D E F G In

# Index

## NOTES