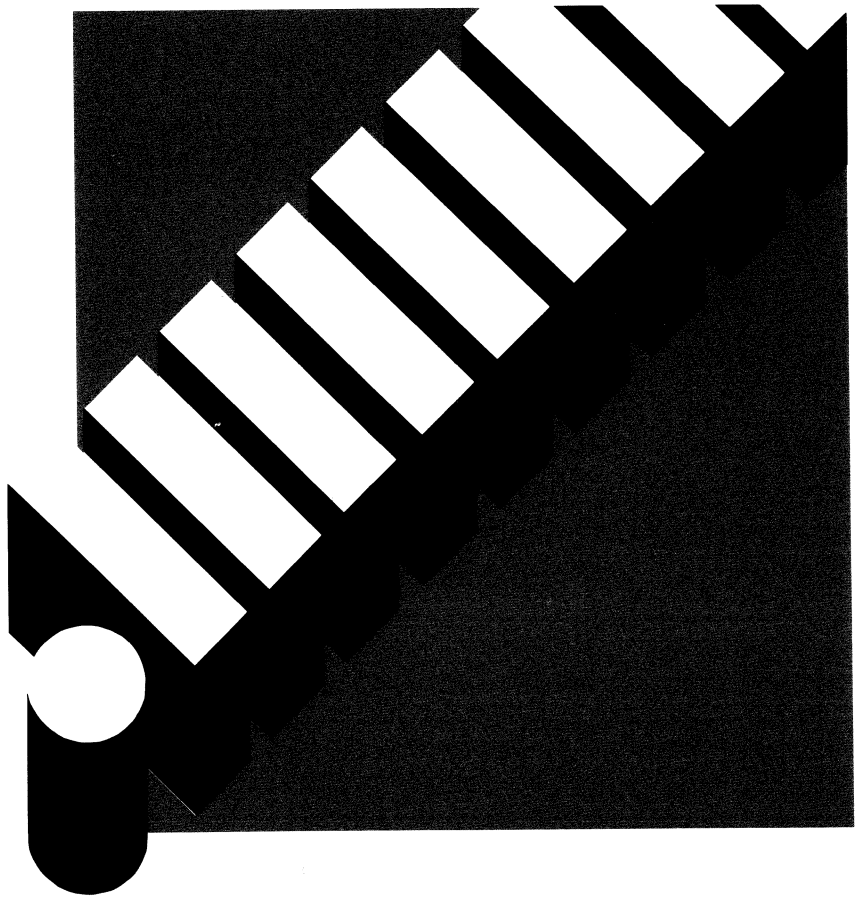
ASHTON-TATE®

LANGUAGE REFERENCE



Language Reference

Language Reference

Copyright © Ashton-Tate Corporation 1988, 1990
All Rights Reserved.

Use of the software contained in this package has been provided under a Software License Agreement. Please read it thoroughly. In summary, you may not make copies of the software except as specifically permitted under the Software License Agreement. You may use the software only on a single terminal or a single workstation of a computer (or its replacement): accordingly, you must license a separate copy for each terminal or workstation where you want to use the software, or use the multi-user version on the permitted number of workstations as set forth in the multi-user license agreement.

Note: Unauthorized use of the software or of the related materials can result in civil damages and criminal penalties.

The software and related materials in this package are licensed with a Limited 90-Day Warranty. **Other than the limited warranties that are expressly stated therein, Ashton-Tate makes no other warranty, express or implied, to you or any other person or entity. We will not be liable for incidental, consequential or other similar damages. In no event will our liability for any damages ever exceed the price paid for the license to use the software, regardless of any form of the claim. You may have other rights which vary from state to state, or country to country.**

Ashton-Tate, the Ashton-Tate logo, dBASE, dBASE II, dBASE III, dBASE III PLUS, dBASE IV, Framework II, Framework III, and RapidFile are registered trademarks of Ashton-Tate Corporation.

CHART-MASTER is a registered trademark of Ashton-Tate Corporation and is used under license from Chartmasters, Inc. Chartmasters is a registered trademark of Chartmasters, Inc.

General Notice: Other product names used herein are for identification purposes only and may be trademarks of their respective companies.

5/90

Contents

Introduction **i-1**
Chapter 1: Essentials i-1
Chapter 2: Commands i-1
Chapter 3: SET Commands i-1
Chapter 4: Functions i-2
Chapter 5: System Memory Variables i-2
Chapter 6: SQL Commands i-2
Chapter 7: SQL Catalogs i-2
Appendixes i-2
Index i-2

Essentials

Essentials **1-1**
About This Chapter 1-1
dBASE IV Language Components 1-1
Dot Prompt Interface 1-2
SQL Prompt Interface 1-3
Filenames and Aliases 1-4
Programs and Procedures 1-7
Using Commands 1-9
Syntax 1-9
Expressions 1-11
Data Types 1-12
Operators 1-14
Precedence of Operators 1-15
Using SET Commands 1-16
Using Functions 1-16
User-Defined Functions 1-17
What is a UDF? 1-17

Example of a UDF 1-18
Limitations on UDFs 1-18
Using System Memory Variables 1-19

Commands

Commands 2-1
?/?? 2-1
??? 2-6
@ 2-10
@...CLEAR 2-20
@...FILL 2-21
@...TO 2-22
ACCEPT 2-24
ACTIVATE MENU 2-25
ACTIVATE POPUP 2-26
ACTIVATE SCREEN 2-27
ACTIVATE WINDOW 2-28
APPEND 2-29
APPEND FROM 2-31
APPEND FROM ARRAY 2-35
APPEND MEMO 2-37
ASSIST 2-39
AVERAGE 2-40
BEGIN/END TRANSACTION 2-41
BROWSE 2-45
CALCULATE 2-50
CALL 2-53
CANCEL 2-54
CHANGE 2-55
CLEAR 2-56
CLOSE 2-58
COMPILE 2-59
CONTINUE 2-62
CONVERT 2-63
COPY 2-65
COPY FILE 2-69
COPY INDEXES 2-70
COPY MEMO 2-72
COPY STRUCTURE 2-73
COPY STRUCTURE EXTENDED 2-75
COPY TAG 2-76

COPY TO ARRAY 2-77
COUNT 2-79
CREATE FROM 2-80
CREATE/MODIFY APPLICATION 2-82
CREATE/MODIFY LABEL 2-84
CREATE/MODIFY QUERY/VIEW 2-86
CREATE/MODIFY REPORT 2-88
CREATE/MODIFY SCREEN 2-90
CREATE/MODIFY STRUCTURE 2-92
CREATE VIEW FROM ENVIRONMENT 2-96
DEACTIVATE MENU 2-98
DEACTIVATE POPUP 2-99
DEACTIVATE WINDOW 2-100
DEBUG 2-101
DECLARE 2-104
DEFINE BAR 2-107
DEFINE BOX 2-109
DEFINE MENU 2-111
DEFINE PAD 2-112
DEFINE POPUP 2-114
DEFINE WINDOW 2-116
DELETE 2-118
DELETE FILE 2-119
DELETE TAG 2-120
DIR 2-122
DISPLAY 2-124
DO 2-125
DO CASE 2-129
DO WHILE 2-131
EDIT 2-133
EJECT 2-136
EJECT PAGE 2-137
ERASE/DELETE FILE 2-139
EXPORT 2-140
FIND 2-141
FUNCTION 2-144
GO/GOTO 2-148
HELP 2-150
IF 2-151
IMPORT 2-153
INDEX 2-154
INPUT 2-160
INSERT 2-161

JOIN 2-163
KEYBOARD 2-165
LABEL FORM 2-167
LIST/DISPLAY 2-169
LIST/DISPLAY FILES 2-171
LIST/DISPLAY HISTORY 2-172
LIST/DISPLAY MEMORY 2-173
LIST/DISPLAY STATUS 2-176
LIST/DISPLAY STRUCTURE 2-178
LIST/DISPLAY USERS 2-180
LOAD 2-181
LOCATE 2-184
LOGOUT 2-186
MODIFY COMMAND/FILE 2-187
MOVE WINDOW 2-190
NOTE 2-191
ON ERROR/ESCAPE/KEY 2-192
ON PAD 2-197
ON PAGE 2-198
ON READERROR 2-200
ON SELECTION PAD 2-201
ON SELECTION POPUP 2-202
PACK 2-204
PARAMETERS 2-205
PLAY MACRO 2-206
PRINTJOB/ENDPRINTJOB 2-207
PRIVATE 2-209
PROCEDURE 2-210
PROTECT 2-213
PUBLIC 2-217
QUIT 2-219
READ 2-220
RECALL 2-222
REINDEX 2-223
RELEASE 2-224
RENAME 2-226
REPLACE 2-228
REPLACE FROM ARRAY 2-230
REPORT FORM 2-232
RESET 2-234
RESTORE 2-235
RESTORE MACROS 2-236
RESTORE SCREEN 2-237

RESTORE WINDOW 2-238
RESUME 2-239
RETRY 2-240
RETURN 2-241
ROLLBACK 2-242
RUN 2-244
SAVE 2-246
SAVE MACROS 2-247
SAVE SCREEN 2-248
SAVE WINDOW 2-249
SCAN 2-250
SEEK 2-252
SELECT 2-254
SHOW MENU 2-256
SHOW POPUP 2-257
SKIP 2-258
SORT 2-260
STORE 2-263
SUM 2-266
SUSPEND 2-267
TEXT 2-268
TOTAL 2-269
TYPE 2-271
UNLOCK 2-272
UPDATE 2-273
USE 2-275
WAIT 2-279
ZAP 2-280

Set Commands

SET Commands 3-1
SET 3-1
SET ALTERNATE 3-2
SET AUTOSAVE 3-4
SET BELL 3-5
SET BLOCKSIZE 3-6
SET BORDER 3-7
SET CARRY 3-10
SET CATALOG 3-11
SET CENTURY 3-15
SET CLOCK 3-16

SET COLOR 3-17
SET CONFIRM 3-26
SET CONSOLE 3-27
SET CURRENCY 3-28
SET CURRENCY LEFT/RIGHT 3-29
SET CURSOR 3-30
SET DATE 3-31
SET DBTRAP 3-32
SET DEBUG 3-34
SET DECIMALS 3-35
SET DEFAULT 3-36
SET DELETED 3-37
SET DELIMITERS 3-38
SET DESIGN 3-40
SET DEVELOPMENT 3-41
SET DEVICE 3-42
SET DIRECTORY 3-43
SET DISPLAY 3-45
SET ECHO 3-46
SET ENCRYPTION 3-47
SET ESCAPE 3-48
SET EXACT 3-49
SET EXCLUSIVE 3-51
SET FIELDS 3-52
SET FILTER 3-56
SET FORMAT 3-58
SET FULLPATH 3-60
SET FUNCTION 3-61
SET HEADINGS 3-63
SET HELP 3-64
SET HISTORY 3-65
SET HOURS 3-66
SET INDEX 3-67
SET INSTRUCT 3-70
SET INTENSITY 3-71
SET LOCK 3-72
SET MARGIN 3-73
SET MARK 3-75
SET MEMOWIDTH 3-76
SET MESSAGE 3-77
SET NEAR 3-78
SET ODOMETER 3-80
SET ORDER 3-81

SET PATH 3-83
SET PAUSE 3-84
SET POINT 3-85
SET PRECISION 3-86
SET PRINTER 3-87
SET PROCEDURE 3-91
SET REFRESH 3-93
SET RELATION 3-95
SET REPROCESS 3-99
SET SAFETY 3-100
SET SCOREBOARD 3-101
SET SEPARATOR 3-102
SET SKIP 3-103
SET SPACE 3-105
SET SQL 3-106
SET STATUS 3-107
SET STEP 3-108
SET TALK 3-109
SET TITLE 3-110
SET TRAP 3-111
SET TYPEAHEAD 3-112
SET UNIQUE 3-113
SET VIEW 3-115
SET WINDOW 3-117

Functions

Functions 4-1
& 4-1
ABS() 4-3
ACCESS() 4-4
ACOS() 4-6
ALIAS() 4-7
ASC() 4-8
ASIN() 4-9
AT() 4-10
ATAN() 4-11
ATN2() 4-12
BAR() 4-13
BOF() 4-14
CALL() 4-15
CDOW() 4-16

CEILING() 4-17
CERROR() 4-18
CHANGE() 4-20
CHR() 4-21
CMONTH() 4-23
COL() 4-24
COMPLETED() 4-25
COS() 4-26
CTOD() 4-27
DATE() 4-28
DAY() 4-29
DBF() 4-30
DELETED() 4-31
DIFFERENCE() 4-32
DISKSPACE() 4-33
DMY() 4-34
DOW() 4-35
DTOC() 4-36
DTOR() 4-37
DTOS() 4-38
EOF() 4-39
ERROR() 4-40
EXP() 4-41
FIELD() 4-42
FILE() 4-43
FIXED() 4-44
FKLABEL() 4-45
FKMAX() 4-46
FLOAT() 4-47
FLOCK() 4-48
FLOOR() 4-50
FOUND() 4-51
FV() 4-52
GETENV() 4-53
IIF() 4-54
INKEY() 4-56
INT() 4-61
ISALPHA() 4-62
ISCOLOR() 4-63
ISLOWER() 4-64
ISMARKED() 4-65
ISUPPER() 4-66
KEY() 4-67

LASTKEY() 4-68
LEFT() 4-69
LEN() 4-70
LIKE() 4-71
LINENO() 4-72
LKSYS() 4-73
LOCK() 4-75
LOG() 4-76
LOG10() 4-77
LOOKUP() 4-78
LOWER() 4-79
LTRIM() 4-80
LUPDATE() 4-81
MAX() 4-82
MDX() 4-83
MDY() 4-84
MEMLINES() 4-85
MEMORY() 4-86
MENU() 4-87
MESSAGE() 4-88
MIN() 4-89
MLINE() 4-90
MOD() 4-91
MONTH() 4-92
NDX() 4-93
NETWORK() 4-94
ORDER() 4-95
OS() 4-96
PAD() 4-97
PAYMENT() 4-98
PCOL() 4-99
PI() 4-100
POPUP() 4-101
PRINTSTATUS() 4-102
PROGRAM() 4-103
PROMPT() 4-104
PROW() 4-105
PV() 4-106
RAND() 4-107
READKEY() 4-108
RECCOUNT() 4-110
RECNO() 4-111
RECSIZE() 4-112

REPLICATE() 4-113
RIGHT() 4-114
RLOCK() 4-115
ROLLBACK() 4-116
ROUND() 4-117
ROW() 4-118
RTOD() 4-119
RTRIM() 4-120
RUN() 4-121
SEEK() 4-122
SELECT() 4-124
SET() 4-125
SIGN() 4-128
SIN() 4-129
SOUNDEX() 4-130
SPACE() 4-132
SQRT() 4-133
STR() 4-134
STUFF() 4-135
SUBSTR() 4-136
TAG() 4-137
TAN() 4-138
TIME() 4-139
TRANSFORM() 4-140
TRIM() 4-141
TYPE() 4-142
UPPER() 4-144
USER() 4-145
VAL() 4-146
VARREAD() 4-147
VERSION() 4-149
WINDOW() 4-150
YEAR() 4-151

System Memory Variables

System Memory Variables 5-1
_alignment 5-1
_box 5-3
_indent 5-5
_lmargin 5-6
_padvance 5-8

—pageno 5-10
—pbpage 5-12
—pcolno 5-14
—pcopies 5-16
—pdriver 5-18
—pecode 5-20
—peject 5-21
—pepage 5-22
—pform 5-24
—plength 5-25
—plineno 5-26
—ploffset 5-27
—ppitch 5-28
—pquality 5-30
—pscode 5-31
—pspacing 5-33
—pwait 5-34
—rmargin 5-35
—tabs 5-37
—wrap 5-38

SQL Commands

SQL Commands 6-1
Symbols and Conventions 6-1
Reserved Words 6-2
Classes of Commands 6-3
Creation/Startup of SQL Databases 6-3
Creation/Modification of Objects 6-3
Database Security 6-3
Data Definition 6-4
Deletion of Objects 6-4
Embedded SQL 6-5
Query and Update of Data 6-5
Utility 6-5
ALTER TABLE 6-6
CLOSE 6-9
CREATE DATABASE 6-11
CREATE INDEX 6-13
CREATE SYNONYM 6-16
CREATE TABLE 6-18
CREATE VIEW 6-21

DBCHECK 6-26
DBDEFINE 6-28
DECLARE CURSOR 6-31
DELETE 6-34
DROP DATABASE 6-38
DROP INDEX 6-39
DROP SYNONYM 6-40
DROP TABLE 6-41
DROP VIEW 6-42
FETCH 6-43
GRANT 6-46
INSERT 6-50
LOAD DATA 6-53
OPEN 6-56
REVOKE 6-58
ROLLBACK 6-61
RUNSTATS 6-63
SELECT 6-65
SHOW DATABASE 6-84
START DATABASE 6-85
STOP DATABASE 6-87
UNLOAD DATA 6-88
UPDATE 6-91

SQL Catalogs

SQL Catalogs 7-1
Updating the Catalog Tables 7-3
Description of Catalog Tables 7-4
Sysauth Table 7-4
Syscolau Table 7-5
Syscols Table 7-5
Sysdbs Table 7-6
Sysidxs Table 7-7
Syskeys Table 7-7
Syssyns Table 7-8
Systabls Table 7-8
Systimes Table 7-9
Sysvdeps Table 7-9
Sysviews Table 7-9

Appendixes

Error Messages **A-1**
dBASE Error Messages A-1
SQL Error Messages A-33

dBASE IV Specifications **B-1**
Database File B-1
Index File B-1
Field Sizes B-1
Arrays B-1
Multi-user Procedures B-2
File Operations B-2
Numeric Accuracy B-2
Memory Variables B-2
Run-Time Symbols B-2
Compile-Time Symbols B-3
Capacities B-3
Word Wrap Editor B-3
Forms B-3
Reports B-3
Labels B-3
QBE B-4
SQL B-4
Applications Generator B-4
Miscellaneous Capacities B-4

Sample Files **C-1**
Client.dbf C-1
Transact.dbf C-3
Stock.dbf C-4
Menus.prg C-5

File Extensions **D-1**
File Extensions Used by dBASE IV D-1
File Relations and Compatibilities D-4

Structure of a Database (.dbf) File **E-1**
Database Header and Records E-1
Database Header Structure E-2
Database Records E-3
Memo Fields and the .dbt File E-4

dBASE IV Printer Drivers **F-1**

ASCII Chart **G-1**

dBASE Commands and Functions Allowed in SQL Mode **H-1**

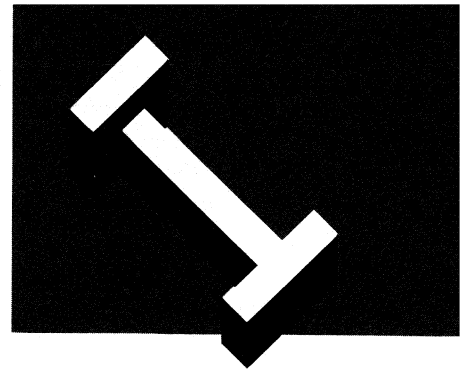
dBASE Commands Allowed in SQL Mode H-1

dBASE Functions Allowed in SQL Mode H-3

Index

Index **X-1**

Introduction



Language Reference is an encyclopedia of dBASE IV® commands, functions, and system memory variables. 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*.

This manual is organized into seven chapters, eight 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 IV 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: SQL Commands

This chapter provides syntax, description, and examples for each SQL command.

Chapter 7: SQL Catalogs

This chapter describes SQL catalog tables.

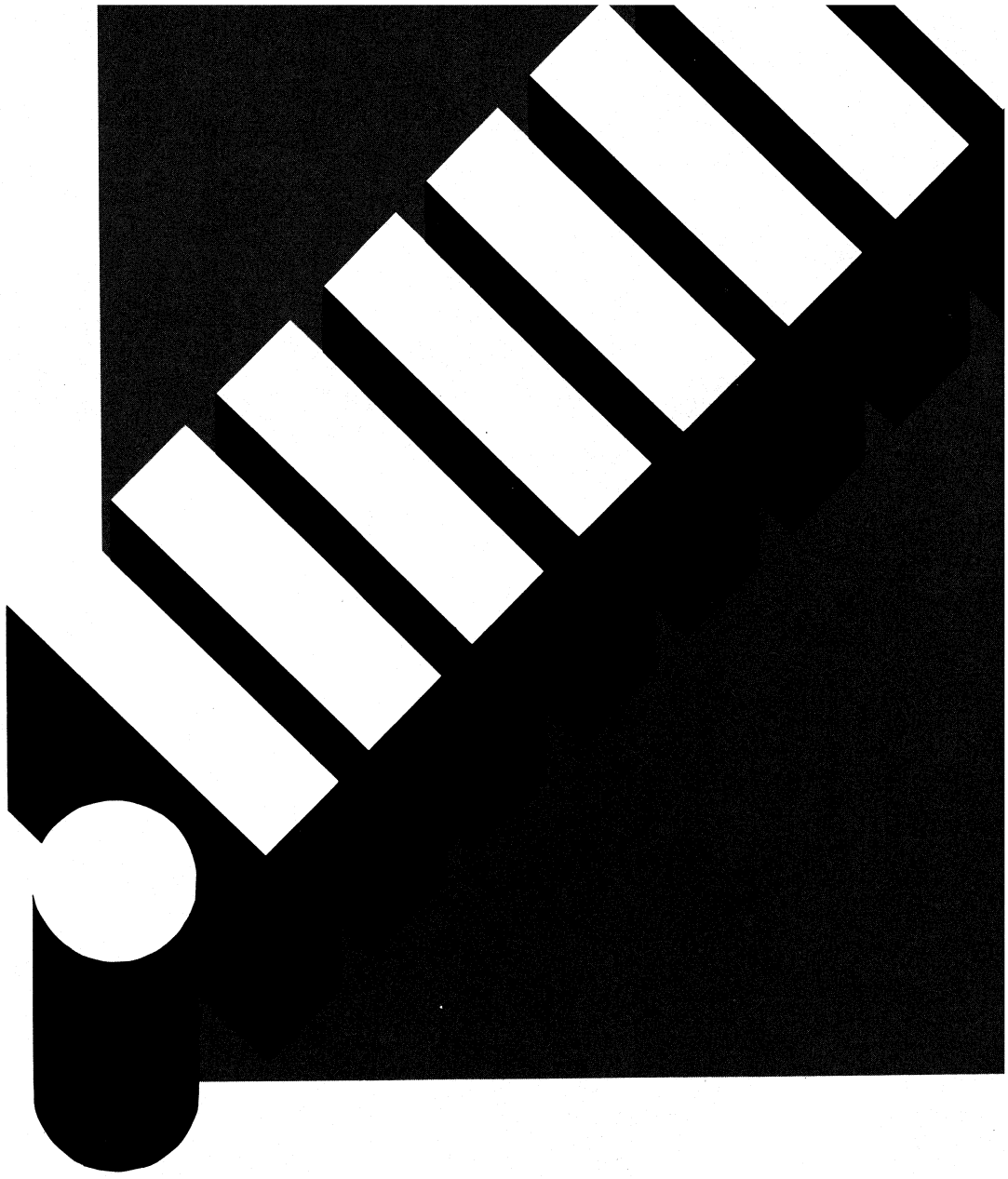
Appendixes

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
- dBASE commands and functions that can be used with SQL.

Index

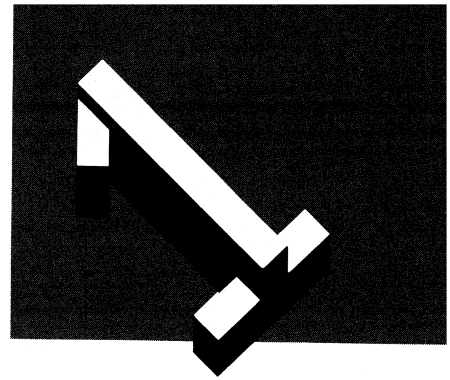
This is a convenient cross-reference to all the concepts, commands, functions, and variables discussed in the *Language Reference* manual.



i 1 2 3 4 5 6 7 A B

C D E F G H 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*, once 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 several ways:
  1. Some functions, like PI(), always return a constant value.
  2. Some functions are adjectives or adverbs that modify a data item. For example, LOWER() converts uppercase letters to lowercase.
  3. Other functions hold a dBASE IV value or condition that you can query. The function EOF() is set to true (.T.) when you are at the end of a database file, and the command:

```
. ? EOF()
```

     returns

```
.T.
```

  4. 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*.
  5. Other functions, like LOOKUP(), SEEK(), RLOCK(), and FLOCK(), perform an action and return a value.
- 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. They are like memory variables in that they can be public or private.

In this manual, commands are printed in uppercase, such as LIST. SET commands are also uppercase, and always begin with the word SET, such as SET DEVICE. Functions are also printed in uppercase, but end with parentheses, such as FOUND(). System memory variables are lowercase, 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

The interactive mode of dBASE IV, 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.

Entering Commands

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 uppercase, lowercase, 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 in the maximum characters 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 *Getting Started*).

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 edit the stored command, and you can run the command by pressing ↵. Use DISPLAY HISTORY and LIST HISTORY to view more than one command at a time.

SQL Prompt Interface

The interactive SQL mode lets you enter a SQL command line in much the same way as you enter a dBASE command. At the dot prompt, you can switch to SQL mode by entering SET SQL ON. The SQL prompt, SQL., appears. To re-enter dBASE mode and redisplay the dot prompt, enter SET SQL OFF at the SQL prompt.

You enter a SQL command directly at the SQL prompt or in an editing window, just as in dBASE mode. You also can use the history buffer to re-enter SQL commands. The conditions for entering SQL commands are also the same as for dBASE commands, except that you may not abbreviate SQL command keywords.

While in SQL mode, you can use most dBASE commands and functions to supplement the operation of SQL commands. These commands and functions are listed in Appendix H.

For example, you can use dBASE commands to process and format SQL results for printing reports and labels and to define the SQL environment. You can use dBASE functions to influence how SQL commands are processed and how results are displayed.

Before entering SQL commands for tables, views, and other SQL objects, you must start a database using the SQL START DATABASE command. For information about SQL command usage, refer to Chapter 6.

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 (new to dBASE IV). 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.

Using indirect references to files is an important new feature of dBASE IV. An indirect reference is a character expression that evaluates to a filename and can be used anywhere you are asked to provide a filename. You must use an operator in the expression (usually parentheses) 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)
```

dBASE IV interprets certain characters that you might want to include in a filename, such as () and &. To include these characters in a filename, use an indirect reference:

```
. CREATE "TEST()"
```

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 can be opened automatically later.

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, the commands listed previously 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. If you 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\MYDATA, 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. The .qbe, .upd, .frm, .lbl, .vue, and .cat files contain filenames with drive and path names only 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.

NOTE

You can use an indirect reference to an alias, except when the alias is used as a prefix for a field name. If you include an operator (usually parentheses) in the alias character string, dBASE IV knows that the string is an expression, not the literal alias. For example,

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

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

```
. ? RECCOUNT("Client")
```

returns the number of records in the Client file, which is open in another work area.

Indirect alias references are similar to using the macro substitution character, but operate much faster.

Programs and Procedures

If you execute the same series of commands repeatedly, you can automate the process by saving the 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 a 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 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 or Control Center.

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. 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, for the first procedure with that name.
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 a 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 the rules for building expressions.

Syntax

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 shown 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 a 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 angle 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 following this section).

<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 act as restrictions within the <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 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 store user input, perform calculations, comparisons, and for 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 any contiguous group of characters.

IN <alias> 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 and the use of an indirect reference 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 a *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. 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" (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 a function to convert a numeric variable to character data type before joining it 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 that can be used in an expression: *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

In addition, <memo field name> is used in syntax to distinguish variable length fields used to hold large blocks of text. Memo field data is not stored in the database (.dbf) file, but in a separate .dbt file associated with the .dbf file.

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.

Because type N numbers contain a decimal representation, they are not subject to rounding errors. (They are subject to errors if you exceed the numeric accuracy, however. See SET PRECISION in Chapter 3 for a discussion of numeric accuracy with type N numbers.) Type N numbers 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. 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 .6E + 23.

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 *Getting Started* 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.

Enter a blank date in any of the following ways:

```
{ }
{ / / }
CTOD (" / / ")
```

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. 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, or date expressions. Both expressions you use in a relational operation must be of the same type. Logical comparisons of character strings are affected by the SET EXACT command.

<	Less than
>	Greater than
=	Equal to
<> or #	Not equal to
<= or =<	Less than or equal to
>= or =>	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
=	Equal to
<> or #	Not equal to

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 of the string preceding the operator 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 the dBASE IV system parameters from the dot prompt or from within program and procedure files. Chapter 3 lists and annotates all the dBASE IV SET commands and their default settings.

You can temporarily change any of the SET command settings from the dot prompt, or by using the full-screen menu available by typing SET. These settings are only active until you reset them with another SET command, or until you QUIT. You can change some default settings, which are the settings that appear when you start dBASE IV, by modifying the Config.db file. See *Getting Started* for information on Config.db settings.

There are two common forms for SET commands:

SET <parameter> ON/OFF

or

SET <parameter> TO <expression>

Some dBASE III PLUS SET commands, such as SET DOHISTORY, SET FIXED, and SET MENU, are not used by dBASE IV. For compatibility with dBASE III PLUS programs, dBASE IV does not return an error message when one of these SET commands is in a program or is entered at the dot prompt. These SET commands have no effect because the new design of dBASE IV removed the need for these settings.

Using Functions

Functions perform specialized operations that augment and enhance the dBASE IV commands. Functions return a value. Some functions evaluate or convert data, and some perform an action.

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

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 does not compile the command line 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 processing time.

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

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

If you do this, 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 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.

TIP
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 keyboard macros, in which you store a series of keystrokes.

User-Defined Functions

dBASE IV allows you to create your own functions. With a user-defined function (UDF), you can enhance the dBASE IV language to perform other operations.

What is a UDF?

A UDF is a special kind of procedure that can be called from within a dBASE IV command line. It begins with the FUNCTION command, and contains commands and an optional parameter list that it uses to return a value. The UDF procedure begins with a FUNCTION command followed by the name you provide for this UDF. The FUNCTION command distinguishes a UDF as a special type of procedure. Although other procedures do not require a RETURN command, you must end the UDF with a RETURN command and values to be returned. See the FUNCTION and RETURN entries in Chapter 2, "Commands," for further information on using these commands.

After you create a UDF, you can call it from a command line just as you would use any other dBASE IV function. The syntax for calling a user-defined function is:

<UDF name>([<parameter list>])

When dBASE IV encounters the UDF name in a command line, it searches for and executes the UDF procedure. Therefore, both the UDF name in the calling command line and the associated FUNCTION command in the procedure must have the same name. The optional parameter list contains expressions that are passed to the UDF. The UDF may contain a PARAMETERS command line that assigns local variable names to each item in the parameter list. The parentheses [()] must follow the UDF name in the calling command line, even if these parentheses do not contain parameters.

Example of a UDF

The following sample code, contained in a hypothetical program named Calendar, illustrates the essential parts of a UDF.

The first part of the program prompts the user to enter a year. The program prints dates and other information, which is not shown in this example. The program must print an item for 29 February only if the year is a leap year, because this date only occurs in leap years. If the year is not a leap year, the program must print blanks in this area. The command line

@ 19,45 SAY IIF(LEAP(year), "29 February", SPACE(11))

prints either a line item for 29 February or blanks, depending on the value returned by the UDF named LEAP(). This command passes a parameter, year, which is the year that the user entered, to the UDF. If the UDF returns a logical true, the text "29 February" prints. If the UDF returns a logical false, the area is blanked out with 11 spaces and no text prints. Check the usage of the IIF() function in Chapter 4, "Functions."

The UDF that determines if the year is a leap year begins with the FUNCTION command and ends with the RETURN command. Note that this procedure is contained in Calendar.prg, but it could also be in any other open procedure file. The UDF name, LEAP(), corresponds to the calling statement in the @ command line above. The procedure creates one local variable, yr, for the parameter, year, that was passed by the @ command line. The next line stores a logical true to the variable isleap if the year is a leap year, and a logical false if the year is not a leap year. Check the MOD() function in Chapter 4, "Functions," to see how MOD() determines this. The RETURN command line returns the true or false value to the @ command line that called the UDF.

```
* Calendar.prg
*
SET TALK OFF
year = 0
@ 12,12 SAY "Enter the new fiscal year: " GET year PICTURE "9999"
READ
.
.
.
@ 19,45 SAY IIF(LEAP(year), "29 February", SPACE(11))
.
.
.
FUNCTION LEAP
PARAMETERS yr
isleap = ((MOD(yr,4) = 0) .AND. (MOD(yr,100) <> 0)) .OR. (MOD(yr,400) = 0)
RETURN isleap
```

Limitations on UDFs

Two types of limitations affect how you use UDFs: general UDF rules and restrictions on intervening commands. General UDF rules apply when you create or run a UDF. Restrictions on intervening commands apply to format files and ON command processing, as well as to UDFs.

General UDF Rules

The UDF name cannot be the same as an existing dBASE IV function or command. If a UDF has the same name as an existing dBASE IV function, the dBASE IV function executes, not the UDF. For example, do not name your user-defined function EXP, because EXP() is a dBASE IV function.

You can nest UDFs, so that one UDF calls another UDF. The maximum number of UDFs that can be nested is 32, but the upper limit really depends on the complexity of each of these nested UDFs.

You cannot use UDFs in a SQL command, nor can you use SQL commands in a UDF. The UDF may be called from a SQL program (.prs) file, however, as long as it not called from a SQL command, does not contain SQL commands, and does not contain any dBASE commands that are prohibited in SQL mode. See Appendix H for the list of dBASE commands permitted in SQL mode.

Restrictions on Intervening Commands

UDFs often interrupt a dBASE IV command in order to execute other commands. The commands that are executed during the interruption are called intervening commands. SET FORMAT and some of the ON commands also interrupt commands in order to execute intervening commands. All intervening commands, whether called by ON, SET FORMAT, or UDFs, are subject to certain restrictions.

In intervening commands, you cannot recursively APPEND, BROWSE, or EDIT the currently active work area. Therefore, you cannot BROWSE a database file containing a calculated field that calls a UDF, if the UDF will try another BROWSE (or an APPEND or EDIT) of the same database file.

You also cannot use LIST and DISPLAY recursively in intervening commands, even if the commands are LISTing or DISPLAYing records in another work area. Therefore, a UDF called from a LIST command cannot include another LIST command (or a DISPLAY command).

Other restrictions on intervening commands are affected by the DBTRAP setting. DBTRAP provides a net of protection against program errors that may give you unexpected, and undesired, results. Although DBTRAP can be reset with the SET DBTRAP command or by changing DBTRAP in the Config.db file, you should not change this setting unless you are aware of the possible results, and are either sure these results will not affect your program or are providing other safeguards if an error occurs.

See Chapter 3, "SET Commands," for further information on the restrictions and uses of SET DBTRAP.

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 printjobs, such as form feeds and the number of copies printed
- The appearance of paragraphs in reports and memo fields, 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
- Defaults of the word wrap editor, such as width of tabs.

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 is not positioned with row and column coordinates, but its destination 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 and 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.

NOTE
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 appear.

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

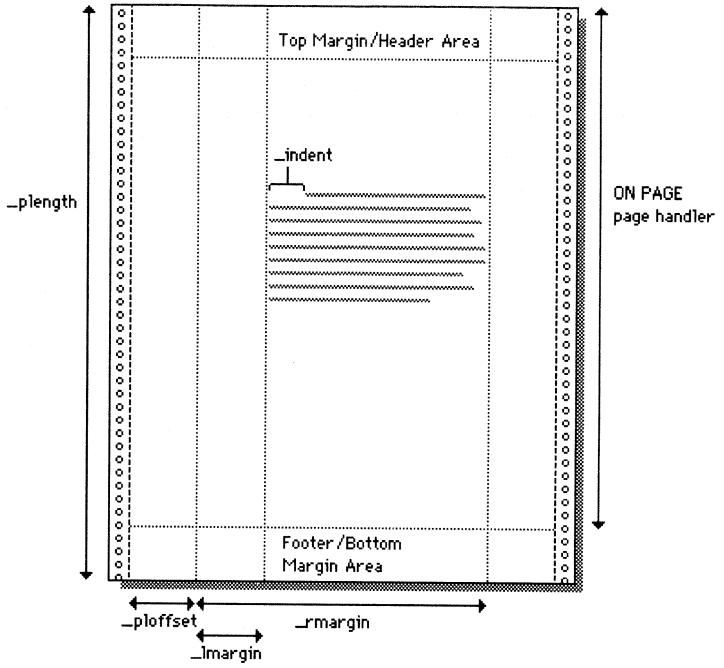


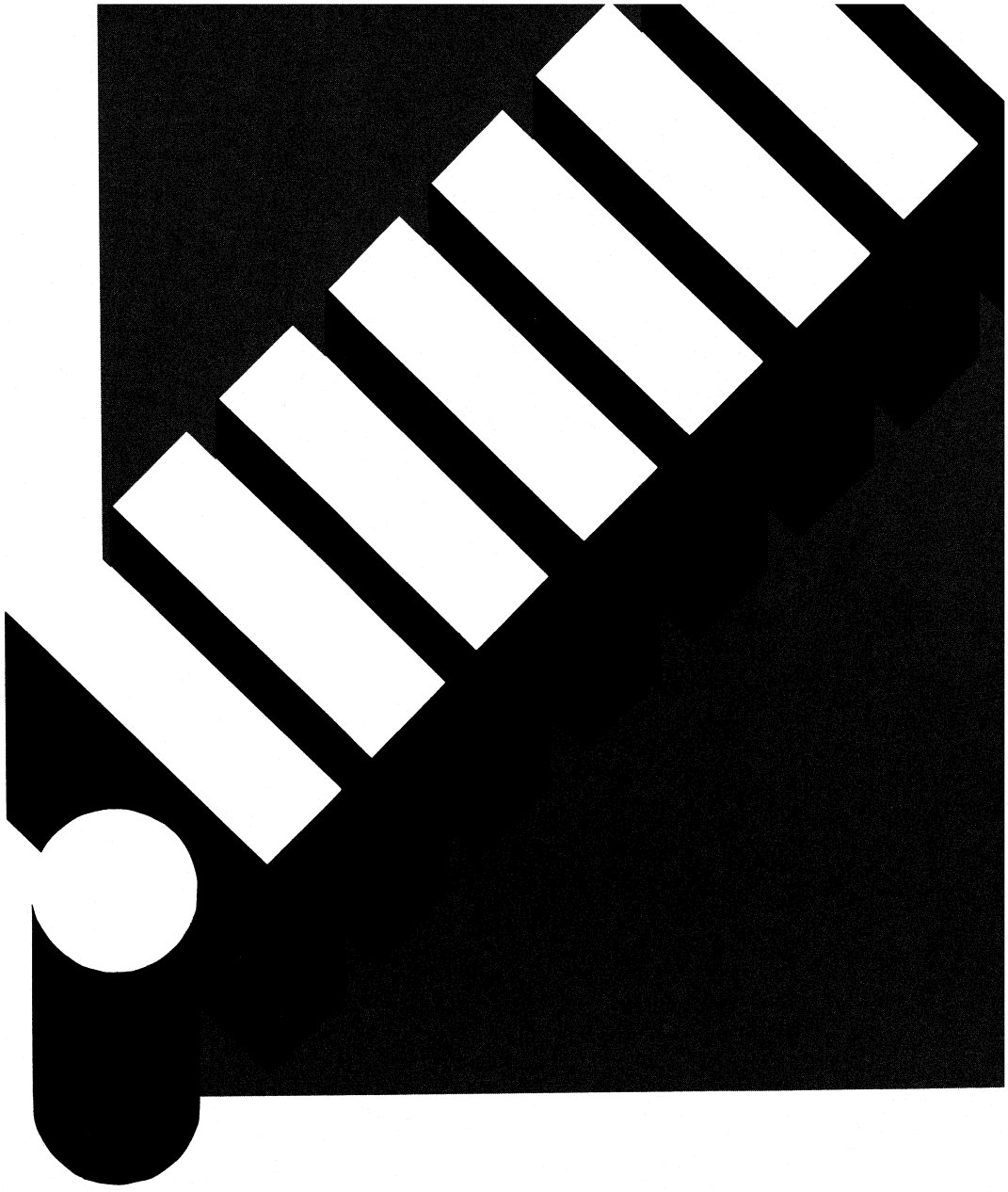
Figure 1-1 Page layout for typical printjob

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."

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*, which has a .prf extension. To change the default settings in this file, refer to *Using the Menu System*. You can also create additional customized print form files. These files normally have 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.

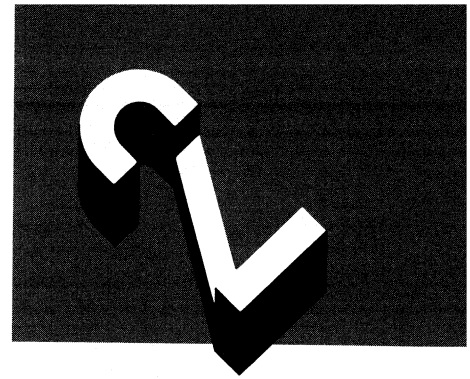
Commands



i 1 2 3 4 5 6 7 A B

C D E F G H In

Commands



?/??

?/?? evaluates and displays the value of one or more expressions with optional formatting, styling, and positioning. The output of ?/?? can go to the console, the printer, or to an alternate file, exclusively or in combinations.

Syntax

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

Usage

If SET CONSOLE is ON, the output of the ? and ?? commands is sent to the console. If SET PRINTER is ON, the output of the ? and ?? commands is sent to the printer. If an alternate file has been set with SET ALTERNATE TO <filename>, and SET ALTERNATE is ON, then output of the ? and ?? commands is also sent to the alternate file. Any or all of the console, printer, and alternate file devices may be SET ON at any time.

? outputs a carriage return and line feed before displaying the results of the expression list.

?? displays the expression list starting at the current output position.

You may specify as many expressions with optional PICTURE, FUNCTION, AT, and STYLE clauses to the ?/?? commands as will fit on the 1,024-character command line.

If you need to display more expressions than can fit on the command line, you may use the trailing comma to specify that another command is pending. A trailing comma indicates that complete output of all expressions in the current ? or ?? command should be delayed until execution of the next ? or ?? command that does not end with a trailing comma. For example:

```
* Print report detail
?? date_trans          AT 0,
?? part_id             AT 10,
?? Goods->part_name    AT 21,
?? part_qty            AT 53 PICTURE "999",
?? Goods->price        AT 58 PICTURE "99,999.99",
* Extend price
tot_price = ROUND(part_qty * Goods->price,2)
? tot_price            AT 70 PICTURE "99,999.99"
?
```

Options

The PICTURE, FUNCTION, AT, and STYLE options let you customize the appearance of output. Note that the STYLE clause is ignored when ? or ?? outputs to an alternate file.

PICTURE templates and FUNCTION clauses format the output. All of the PICTURE template symbols and functions used for output work with the ?/?? command. See the @ command for descriptions.

Three functions (V, H, and ;) are used only with the ?/?? command.

The V<n> function specifies that ?/?? command expressions are displayed in a vertical column with a width of <n> characters. Note that without @V, the width of memo fields defaults to the number of characters established by SET MEMOWIDTH. If you vertically stretch a field with @V, the text inside it can wrap. Non-memo fields, however, are 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.

Using the V function with a 0 (zero) parameter has no effect on non-memo fields. On a memo field, however, V0 causes the lines of the field to be output just as they appeared in the editor. When you are printing fields stretched by the V function, these fields will finish printing, and a line break will appear, before a V0 field begins to print.

The H function is used for memo fields only when the __wrap system memory variable is true (.T.). When you use the H function, memo fields word wrap using the system memory variables __lmargin and __rmargin. The H function is used only with memo fields.

The ; function causes text to wrap at the point where semicolons are encountered in that text. The semicolons in the text do not appear in the output. See Chapter 5, "System Memory Variables," for more information on various formatting controls.

dBASE IV always honors paragraph breaks contained within memo fields.

Some expressions may contain data that does not completely fill the specified PICTURE template; these are called *short fields*. There are four functions that align short fields within their PICTURE templates:

@B Left-aligns data within a template
@I Centers data within a template
@J Right-aligns data within a template
@T Used with alignment functions to trim off blank spaces prior to alignment

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 RIGHT, it displays just after the amount. The L function displays leading zeros in short fields.

If a ?/?? command ends with a comma, the first line of all expressions will be displayed, but subsequent lines of vertically stretching expressions will be kept internally and not output until one of the following events takes place:

1. An ENDPRINTJOB command is executed.
2. A ? command is executed.
3. A ?? command without a trailing comma is executed.

In items 2 and 3 above, a ? command or a ?? command, executed within an ON command such as ON PAGE or ON ESCAPE, will not end output of a vertically-stretched field which started prior to the ON command. But the output of vertically-stretched fields started during an ON command is ended by executing a ?? command without a trailing comma or by executing any ? command.

The optional AT clause allows you to specify the column at which the expression displays. The numeric expression must be between 0 and 255. Use this option to display columns of text which must line up, regardless of the length of the printed text, to the left of the column.

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

The letters you can use in a STYLE clause are:

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

The numbers you can use are 1 through 5, and they activate the user-defined fonts. The starting and ending control codes for these fonts are defined in the Config.db file through the PRINTER <printer int> FONT <font int> statement.

You may combine different styles, and print different text on the same line in different styles.

You may also use the ??? command and the system memory variables _pscode and _pecode to change typestyles. In general, use ?/?? with the STYLE option to change typestyles of individual items, ??? to change styles on a broader basis within a document, and _pscode and _pecode to define the overall typestyle for a document.

Tips

Specify ? without an expression to display a blank line, for example to skip a line in the output. (To double or triple space the output, use the _pspacing system memory variable.) Use ?? without an expression to specify a non-operative function (for example, ON KEY LABEL F2 ??, to disable the **F2** key).

Special Case

Use the B, I, or J functions 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 FUNCTION. The vertical field centering overrides the initial _alignment = "LEFT" setting.

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

If _wrap is set to false (.F.), the H function is ignored when you print a memo field.

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 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 (.T.). With overwrite, only the second line prints. With overstrike, both lines print. Note that overwriting and overstriking apply only to the hard copy of a document; on screen, the document cannot be overwritten or overstruck.

See Also

???, @, PRINTJOB, SET ALTERNATE, SET CONSOLE, SET MEMOWIDTH, SET PRINTER

Chapter 5, "System Memory Variables"

Getting Started

???

??? 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 ({}) and surround the entire string in quote marks (see the Examples).

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}

(continued)

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

ASCII Code	Control Character Specifier
7	{DEL} or {DELETE}
8	{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}
27	{ESC} or {ESCAPE} or {CTRL-[}
28	{CTRL-\}
29	{CTRL-]}
30	{CTRL-^}
127	{BACKSPACE}
403	{CTRL-_}

Tips

To print a left curly brace, enclose it in curly braces, such as ???"{{}". Statements such as, ???CHR(123) or ???"{" will not work.

Special Case

Although it is easier to send output to a PostScript printer from the Control Center or from a program, you can print to a PostScript printer from the dot prompt. Use the following procedure:

1. Enter the USE command for the database file that you want to print.
2. Load the printer driver, for example:

```
. _pdriver="postscri.pr3"
. LIST TO PRINT
. ??? "FF "
```

3. Send some output to the printer to initialize the DLD file.
4. After printing the database file, send an eject command with the "FF " string to eject the last page of output.

If you do not send any output to the printer after initializing the DLD file, the file is erased from the printer's memory. To erase the DLD file manually, enter:

```
. ??? CHR(4)
```

You can use the ??? command to choose resident fonts on the printer. However, proportional fonts will not line up correctly without a program. For example, to choose the default monospace Courier font, enter:

```
. ??? "1FONT "
```

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(), INKEY(), LASTKEY(), READKEY()

Chapter 5, ''System Memory Variables''

Chapter 2 of *Getting Started*

@

@ 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 [REQUIRED] [<low>] [,<high>]]
        [VALID [REQUIRED] <condition>[ERROR <expC>]]
        [WHEN <condition>]
        [DEFAULT <expression>]
        [MESSAGE <expC>]]
    [COLOR [<standard>] [,<enhanced>]]
```

Usage

<row> and <col> are numeric expressions. When you SET DEVICE TO SCREEN, <row> can range from 0 to the height of the screen display, in lines. <col> can range from 0 to 79 columns. <row> and <col> coordinates are always in relation to the upper left corner of the *active* window, whether it is the full CRT screen or a window that you have defined.

When you SET DEVICE TO PRINTER or SET DEVICE TO FILE, the <row> coordinate can range from 0 to 32,767 and the <col> coordinate from 0 to 255.

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.

NOTE
Even though @ does not produce streaming output, it 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 appear.

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, BROWSE [FORMAT], 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. The $ function can be substituted for the current <row> and <column> values. For example, to display a message on incremental rows of the screen, use:

```
. @ $+1, 2 SAY "MESSAGE"
```

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 — Can be used only with format (.fmt) files to assign a preset value to 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. FUNCTION applies to the entire data item, while PICTURE applies to a specified portion of the data item.

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. The message is centered on the bottom line of the screen unless you reposition it with the AT clause of SET MESSAGE.

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 uppercase if the field contains text. If the field is empty, the word "memo" is in lowercase letters.

The <row>, <col> parameters of @...GET used with the WINDOW option refer to local coordinates within the window border.

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

If an @...GET is performed in an active window that is too small to hold the value of the variable being read, the READ command will terminate with the error message, **Position out of window**.

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, 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. The PICTURE and FUNCTION options can each be used twice on any command line, once with SAY and once with GET.

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 prompt **RANGE is <low> to <high> (press SPACE)** appears if you attempt to enter a value out of range. You must press **Spacebar**, then re-enter the value. <low> and <high> are replaced by the word "None" in this message if you did not provide a <low> or <high> expression in the RANGE option. For example, **RANGE is 1 to None (press SPACE)** appears if no <high> expression was provided in the RANGE option and you attempt to enter a 0. You must press **Spacebar**, then enter a value greater than 0.

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.

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.

REQUIRED — The REQUIRED option checks for the VALID clause and the RANGE parameters each time you enter a record, whether you change the record or not. Without REQUIRED, the check occurs only when you change the record.

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 uppercase.
^	Displays numbers in scientific notation.
$	Displays data in currency format.
(	Encloses negative numbers in parentheses.
A	Alphabetic characters only.
B	Left-aligns numeric value in @...SAY only.
C	Displays CR (credit) after a positive number.
D	The current SET DATE format for dates.
E	European date format.

(continued)

Table 2-2 Format functions used in dBASE IV (*continued*)

Function	Description
I	Centers text in @...SAY only.
J	Right-aligns text @...SAY only.
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, 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> (with no space between S and the variable) 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.

NOTE
If the format functions M and ! are both specified in the PICTURE clause of the GET statement, the multiple choice function will override the uppercase function. The text of the multiple choices will not be translated to uppercase when displayed on the screen or when stored to a variable.

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 **Spacebar** to see the remaining choices or until the desired value appears. Press ↵ to select an item and move to the next field.

NOTE
Any data in an input field is lost when the first GET variable choice is displayed in the 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 within a PICTURE template. 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 Template symbols

Symbol	Accepts
9	Allows only digits (0-9) for character data, digits or signs (+ or −) for numeric data.
#	A digit, a space, a period (.), or a sign (+ or −).
A	Alphabetic characters only.
N	Alphabetic or numeric characters, including the underscore (_); no spaces or punctuation.
Y	Y, y, N, or n. Converts y or n to uppercase letters.
L	T, F, Y, or N.
X	Any character.
!	Any character, but converts alphabetic characters to uppercase.
other	Use the R function if you plan to use any other symbol in a template. R allows the literal values to appear in the template, but not in the field contents.

There are four other symbols that may be used in numeric templates. These are described in Table 2-4.

Table 2-4 Numeric template symbols

Symbol	Description
.	Separates integers from decimals with a period (or with another decimal separator indicated by SET POINT).
,	Separates thousands with a comma (or with another character indicated by SET SEPARATOR).
*	Displays asterisks in place of leading zeros.
$	Replaces leading zeros with a dollar sign (or with another currency symbol indicated by SET CURRENCY).

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 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

@...CLEAR, 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.

NOTE
Even though @...TO does not produce streaming output, it may affect the positioning of subsequent output. For example, changing the current cursor position on the screen with an @...TO command affects where a later ?/?? command will appear.

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.

To print a box, use the DEFINE BOX command.

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(), DEFINE BOX, 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, STORE, 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. The DEFINE PAD example in this chapter shows how to set up Main and View.

See Also

DEACTIVATE MENU, DEFINE MENU, DEFINE PAD, 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

@...FILL, ACTIVATE WINDOW, CLEAR, 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.

Example

This command activates a previously defined window named W1:

```
. ACTIVATE WINDOW W1
```

See Also

CLEAR WINDOW, DEACTIVATE WINDOW, DEFINE WINDOW, FUNCTION, 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 ↵ or **Esc** immediately when a new record is presented. 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. If an index is open, they will be in index order. Otherwise, they will be in record 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.

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

@...GET, BROWSE, EDIT, 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 characters are lost, and no numeric data is appended. If the file being APPENDed to has an open index file, the index is automatically updated.

You can test a FOR condition only for those fields that occur in both files with the same name. Further, with a FOR clause, a record in the FROM database file is added to the active database file only if the *resulting* record satisfies the FOR criteria. Because the FOR condition is evaluated for the new record instead of for the FROM database file record, some results may not be what you expect. Following are two examples to illustrate this point.

Example 1:

The value returned for the RECNO() function in the statement APPEND FROM Newdbf FOR RECNO() = 100 is the next available record number in the current database file, not the record number in the FROM database file. In this example, a record is appended only if the current database file has exactly 99 records.

Example 2:

If you attempt to append only those records in the Test database file that are marked for deletion with the statement APPEND FROM Test FOR DELETED(), you will actually be testing the active file (the one that is currently open) for its deleted status. In this example, all records from Test are appended, regardless of their deleted status.

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 Delimited Format ASCII files where the field's delimiter character is other than the standard comma. Character strings are enclosed in double quotation marks, and each line ends with a carriage return and line feed.
- 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).

APPEND FROM

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 into 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 than 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. A row is APPENDed only if the condition would evaluate to true (.T.) against the new record. If the condition is false (.F.), the row is skipped and the next row is processed.

To test for a condition in a column of array elements, specify the database field name corresponding to the array column number, not the column number itself. The Examples section clarifies this situation.

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

Examples

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
```

This example demonstrates how to test for a condition in a column of array elements:

```
DECLARE Temp[5, 11]
Temp[1,1] = "Larry"
Temp[2,1] = "Larry"
Temp[3,1] = "Lenny"
Temp[4,1] = "     "
Temp[5,1] = "Lonny"
USE Travel              && First field is named Firstname
* Append only records from the array that have non-blanks.
APPEND FROM ARRAY Temp FOR Firstname = "L"
4 records added
```

See Also

APPEND, COPY TO ARRAY, DECLARE, STORE

APPEND MEMO

APPEND MEMO imports a file into a named memo field in the current record.

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 Newfile.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...
```

APPEND MEMO

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 gives you access to the dBASE IV Control Center.

Syntax

ASSIST

Usage

ASSIST brings you to the Control Center, from which you can reach the different parts of the dBASE IV 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, COUNT, DECLARE, SET HEADING, SET TALK, SUM

BEGIN/END TRANSACTION

BEGIN TRANSACTION and END TRANSACTION define a transaction — a series of commands treated as a single unit of work. Transaction processing ensures that either all or none of the commands that affect database file contents are processed.

Syntax

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

Usage

BEGIN TRANSACTION creates a log file and starts transaction recording. The transaction remains active until an END TRANSACTION, ROLLBACK, CANCEL, or QUIT is issued. While the transaction is active, dBASE IV logs records that are added or changed, and files that are created.

In a program, issuing a SUSPEND inside a transaction does not interrupt the transaction, but suspends the program that is executing inside the transaction. The RESUME command continues the transaction.

In a program, the ROLLBACK command transfers control to the command immediately following the END TRANSACTION command. Any commands between ROLLBACK and END TRANSACTION are ignored.

The transaction log file is named Translog.log on a stand-alone system, and <computer name>.log on a local area network. <computer name> is the network name of the computer that starts the BEGIN TRANSACTION command. If <computer name> is not available from the network, dBASE IV uses the PROTECT log-in name instead.

By default, the transaction log file is created in the current directory on the current drive. Use the <path name> option to specify a different directory. If the current directory is on a network file server, you can improve network performance by specifying a local workstation directory for the transaction log file.

When used in a program, BEGIN TRANSACTION and END TRANSACTION must be used together, must be in the same procedure, and must appear *together* in the same nested command block in a procedure. Therefore, END TRANSACTION cannot be conditional.

Unlike other structured commands, transactions cannot be nested. You must complete a transaction before you can begin another.

The COMPLETED() and ROLLBACK() functions are used with transaction processing to test the outcome of transactions.

Use COMPLETED() to test whether a transaction has completed successfully. COMPLETED() is true (.T.) unless a transaction is active. CANCEL performs a ROLLBACK and restores the database to its pre-transaction state.

ROLLBACK() tests whether a ROLLBACK command was successful. ROLLBACK() returns true (.T.) by default. If an error occurs during ROLLBACK, it is set to false (.F.) until a subsequent ROLLBACK is successful or you issue a RESET command.

If ROLLBACK is not successful in restoring a group of database files, use the command to restore each file individually.

Commands Not Allowed in Transactions

While a transaction is active, you cannot use the following commands:

CLEAR ALL
CLOSE ALL/DATABASE/INDEX
CONVERT
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 following commands can be used while a transaction is active unless they close open files or overwrite an existing file:

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

If a command attempts to overwrite an existing file or close an open database or index file, an error message appears.

The INDEX command cannot be used if it overwrites an existing .ndx file, .mdx file, or .mdx tag, or if any nonproduction index file is currently open.

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

Tip

File and record locks gained during an active transaction are not released until the transaction is completed. In a multiuser system, you can avoid locking data for extended periods if your programs finish data entry before beginning a transaction. Include only the necessary commands within the transaction, and complete it as soon as possible so that the locks can be released.

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
      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

CHANGE(), 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> [/<column width>] = <expression 1>
[,<field name 2> [/R] [/<column width>]
/ <calculated field name 2> [/column width>] = <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.

Browse will use the active format file (.fmt) when accessed from EDIT via **F2 Data** or from the Control Center.

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 a file has been PROTECTed, you are allowed to read and edit its field information only if your access level gives you the privilege. Otherwise, the message **Unauthorized Access Level** is displayed when you execute BROWSE.

You can change from BROWSE to EDIT by pressing **F2 Data**. You can transfer to query design by selecting **Transfer to Query Design** from the **Exit** menu or by pressing **Shift-F2 Design**.

You can edit all fields except calculated fields or read-only fields. If the changes you make to other fields affect any calculated fields, the calculated fields are recalculated and redisplayed. Calculated fields exist only during a BROWSE session unless they have been listed in a SET FIELDS TO command.

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 in the last BROWSE command to be used again. It instructs the BROWSE command not to reinitialize the BROWSE table, but to reuse 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 prevents you from adding new records to the current file.

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

NOEDIT makes all fields in the table read-only. You can still add records to the file. You can also mark records for deletion.

NODELETE prevents you from deleting records.

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. COMPRESS is ignored in EDIT mode.

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 are then used by BROWSE. Row and column coordinates are ignored because BROWSE always positions fields in a table on the screen.

When a format file is active, data is edited according to the database file attributes and according to the format file specifications. The FORMAT option overrides any FIELDS list specified because the format file determines which fields are used by BROWSE in this case.

LOCK specifies the number of contiguous fields on the left of the screen that do not move when you are scrolling the BROWSE table. <expn> must evaluate to a number between zero and the total number of BROWSE fields. 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. LOCK is ignored in EDIT mode.

WIDTH sets an upper limit on the column widths for all fields in the BROWSE table. <expn> must evaluate to a number between 4 and 99. 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.

For character fields, WIDTH truncates both data and the column heading if necessary. For all other data types, WIDTH truncates only the column heading if necessary (data that would be truncated simply will not appear).

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 does not affect the EDIT command. If you switch to EDIT mode, the cursor will no longer be confined to a particular field. If you switch back to BROWSE mode, FREEZE will again be in effect.

WINDOW activates a window which then defines an area on the screen used by the BROWSE table. The rest of the screen remains intact. You can zoom between full-screen BROWSE and window BROWSE by pressing **F9**. When you exit BROWSE, the window is automatically deactivated. If a window is active when BROWSE is called, BROWSE displays data in that window.

When BROWSE is used in a window, zooming a memo field window with **F9** while editing will deactivate the initial window and leave the screen in full-screen mode. Accessing memo fields from a BROWSE table in an alternate window will deactivate the initial BROWSE window and move the information into the secondary memo field window.

Use FIELDS 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. You can also designate a field as read-only, and construct and name calculated fields to 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 current record, or by determining the longest possible value.

For character fields, the length is set to the length of the result of the calculated field expression. A minimum length of 10 is used for character fields. The length is set to 1 for logical fields, 8 for date fields, and 20 for numeric fields.

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, they are always read-only.

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 99 for character fields, and from 8 to 99 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. <column width> can be used to shrink character fields only. It can never be used to shrink column headings.

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                               && Reposition record pointer.
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:

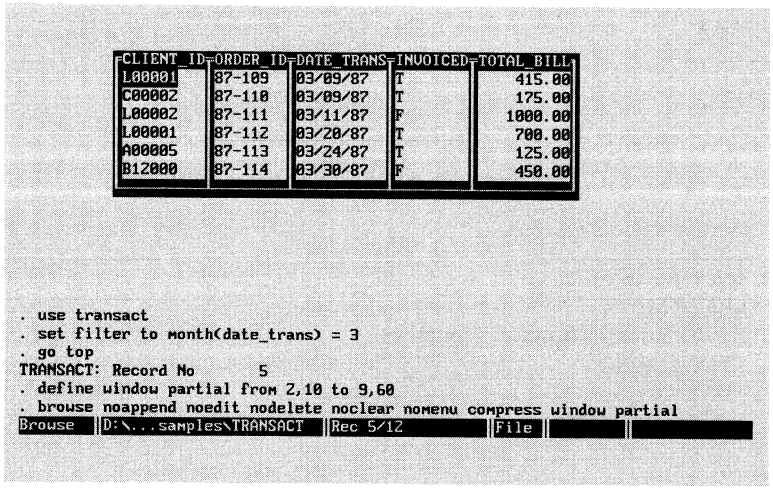


Figure 2-1 BROWSE window

See Also

APPEND, EDIT, PROTECT, SET FIELDS, 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 (fixed) or type F (floating) numbers, depending on the data type of the argument.

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 field. <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.
```

See Also

AVERAGE, COUNT, SUM

CALL

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 (except memo) to the binary program file. All character type memory variables and expressions end with a null (ASCII value of zero).

Refer to the LOAD command in this chapter for details on writing the binary program.

Options

The WITH option is a character expression, field name, memory variable, or array element. You can pass up to seven expressions in the <expression list>. Memory variables, fields, and elements can be of any type. If you pass an expression, you cannot change its value, since expressions are passed as literal values. If you pass a field or a memory variable (or an array element), you can change its value.

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

DEBUG, DO, 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/SCREENS/TYPEAHEAD/
WINDOWS]

Usage

You can enter the CLEAR command without an option, or with a single option. If you enter CLEAR without an option, the command clears the screen.

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 *Getting Started* 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.

SCREENS releases the memory used by screens that you have saved to memory variables using the SAVE SCREENS TO command. CLEAR SCREENS has no effect on the windows addressed by the ACTIVATE SCREENS command, or on screen formatting files (.scr, .fmt, .fmo). CLEAR SCREENS differs from RELEASE SCREENS in that CLEAR SCREENS removes all screen memory variables from memory, whereas RELEASE SCREENS removes only named screens from memory.

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, RELEASE SCREENS, RESTORE WINDOW, SAVE SCREENS, SAVE WINDOW, WAIT

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/INDEXES/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 the keyword for the file type, as specified in the syntax.

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.

CLOSE FORMAT and CLOSE INDEXES close only the format and index files in the current work area. CLOSE INDEXES never closes the production .mdx file.

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
```

See Also

CLEAR ALL, QUIT

COMPILE

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

Syntax

COMPILE <filename>[RUNTIME]

Defaults

NOTE
All dBASE IV version 1.0 object code files must be recompiled in dBASE IV version 1.1 in order to run under version 1.1. Recompilation will in most instances occur automatically. The commands that run a dBASE object file will check the version of the code file and recompile it if it is a version 1.0 code file, and the corresponding source file can be found. dBASE IV will search the current directory and all directories in the dBASE PATH for source files. The recompile will fail if the source file has been renamed or cannot be found, or if the code file was created with the DBLINK utility.

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 programs 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 or warnings that occur from commands that aren't allowed in RunTime use.

COMPILE

When you DO a program file, the source code is compiled into an object code file (if a .dbo file does not already exist), and then the object code is executed. COMPILE allows you to generate the object code file without executing the code.

The dBASE IV internal text editor, which can be accessed from the dot prompt using MODIFY COMMAND or the Control Center, will delete an old .dbo file when a .prg file is modified. A subsequent DO command will recompile the .prg file and create a new .dbo file before executing the procedure.

If SET DEVELOPMENT is ON and you use an external text editor to edit existing program source (.prg) files, 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.

If you do not use the MODIFY COMMAND internal text 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.

Make sure that each source code file has a unique name. If two .prg files share the same name but have different extensions, compiling one may overwrite the other's .dbo file. Also, do not rename a .dbo file. Instead, change the name of the source code file and recompile it.

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

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

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().

Special Case

In a network environment, the database file must be in exclusive use before you issue the CONVERT command.

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 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>]/[[WITH] PRODUCTION]
[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.

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

Usage

The optional [WITH] PRODUCTION keywords work only with dBASE IV database files and copy the production .mdx file associated with the database file. You cannot use the TYPE keyword with the [WITH] PRODUCTION keywords to specify a different file type.

This option provides a way of copying the production index file which cannot be copied any other way.

To use the [WITH] PRODUCTION option, you must have a free work area available.

The key expressions and FOR clauses in the copied production .mdx file must be present in the copied database file. Any indirect file references (aliases) or memory variables used in the .mdx file should be in memory for the copied index to work.

If the TO file is an ASCII text file or a file supported by another software program, specify one of the file type options.

You can use an indirect reference for <filename>. An indirect reference is a character expression that evaluates to a filename, and can be used anywhere you are asked to provide a filename. You must use an operator in the expression (usually parentheses) so that dBASE IV knows that the character string is an expression, not the literal filename. An indirect reference is similar to using the macro substitution character, but operates much faster.

The _dbaselock field in CONVERTed files is not copied to the new file. If PROTECT is in use, fields that the user who is copying the file does not have privileges for are not copied to the new file.

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 copied character by character starting on the left. Each record will 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 in dBASE III PLUS. Any type F (floating) numeric field that you copy using the DBMEMO3 option is converted to type N (fixed).
- RPD — RapidFile data (.rpd) file.
- FW2 — Framework II (.fw2) database.
- SYLK — Multiplan spreadsheet (.slk) format. 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 or a production .mdx file, 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, SET PATH

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.

The .ndx files must be open before you issue the COPY INDEXES command. You may copy a maximum of 10 index (.ndx) files to tags in an .mdx file with one COPY INDEXES command, because up to 10 index files can be open in a work area.

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 INDEXES 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 EXCLUSIVE, SET INDEX, SET ORDER, TAG(), USE

COPY MEMO

COPY MEMO copies the information from a single memo field to an external 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 currently active .dbf and .dbt file if any, but does not copy any records.

Syntax

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

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

If you use the [WITH] PRODUCTION option, COPY STRUCTURE will create an empty .mdx file with all of the index tag expressions that were included in the new .dbf file.

To use the [WITH] PRODUCTION option, you must have a free work area available.

The key expressions and FOR clauses in the copied production .mdx file must be present in the copied database file. Any indirect file references (aliases) or memory variables used in the .mdx file should be in memory for the copied index to work.

This command copies the entire database file structure unless limited by the FIELDS option or SET FIELDS command. The result is another database file with either an identical structure to the first file, or, if you specify the FIELDS option, with the fields specified in the fields list.

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

The _dbaselock field in CONVERTed files is not copied to the new file.

If PROTECT is in use, fields that the user who is copying the file does not have privileges for are not copied to the new file.

Tip

You can COPY STRUCTURE under program control to create temporary database files. You can 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.

COPY STRUCTURE

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. Only those index tags that were selected as "Yes" when the database file structure was created are copied into the FIELD_IDX field. These are single-field tags that have the same tag name as the index field name. Composite indexes are not copied.

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.

The _dbaselock field in CONVERTed files is not copied to the new file. If PROTECT is in use, fields that the user who is copying the file does not have privileges for are not copied to the new file.

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 the dBASE product did not include index flags.

Example

See the CREATE FROM command.

See Also

APPEND FROM, COPY STRUCTURE, 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.

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
 Master index: 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. To copy more than one record to an array, you must DECLARE a two-dimensional array (an array with both rows and columns).

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.).

COPY TO ARRAY

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.
. DECLARE [RECCOUNT(),5]                && Define the array.
. COPY TO ARRAY Records FOR Client_id = "L00001"
    2 records copied
```

If RECCOUNT() exceeds 234 records, the error message **Bad array dimensions** appears. This is because the maximum array dimension is 1170 (5 x 234).

See Also

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

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. COUNT can only act on fields included in a SET FILTER TO command.

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 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.

If any fields in the file created with COPY STRUCTURE EXTENDED have FIELD_IDX = Y, a production .mdx file will be created with the appropriate tags in the new file.

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
. 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/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 objects listed in Table 2-5 with the Applications Generator.

Table 2-5 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.

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 an .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 SET DESIGN is ON, you can display the query design screen from BROWSE or EDIT by pressing **Shift-F2 Design**.

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 determine whether a new .lbo file 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

DO, LABEL FORM, SET CATALOG, SET SAFETY, SET VIEW, _pageno

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 update query (.upd) files that can modify records in the database file. Using a query file, you can activate indexes, perform sorts, and calculate sums.

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 the dBASE product.

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 the dBASE product. 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 generates a file that is automatically given an .frg extension. 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 using the CREATE/MODIFY REPORT command, 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, _pageno

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.

When you create a new format file, it is automatically opened. The next time you enter a command that requires a screen format (APPEND, CHANGE, EDIT, INSERT, OR READ), the new format is used.

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; include additional text, boxes, and lines; define colors of fields, text, boxes, and lines; and select, move, and copy fields, boxes, and blocks of text.

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.

To use a format file, you must USE its associated .dbf file. Then, open the format file with SET FORMAT TO <format filename> (unless you have just created it). If you do not open a format file, the APPEND, CHANGE, EDIT, INSERT, and READ commands use the default entry screen, which looks like the Quick Layout screen.

You can open or close the format file you created at any time. Use SET FORMAT TO <format filename> to open a format file and SET FORMAT TO (with no parameter) to close it. You can also use CLOSE FORMAT to close a format file.

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 the dBASE product, 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, READ, SET CATALOG, SET FORMAT, SET SAFETY, SET VIEW

Using the Menu System contains further information on using the screen designer.

CREATE/MODIFY STRUCTURE

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

Use CREATE to build a structure for a new database file. Use MODIFY STRUCTURE to modify the structure of a previously created database file.

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 specify a drive and directory with the filename, CREATE writes the new database file in the default drive and directory. 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 <filename> 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 can use an indirect reference for <filename>. An indirect reference is a character expression that evaluates to a filename, and can be used anywhere you are asked to provide a filename. You must use an operator in the expression (usually parentheses) so that dBASE IV knows that the character string is an expression, not the literal filename. An indirect reference is similar to using the macro substitution character, but operates much faster.

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 eight bytes. Memo fields are automatically assigned a length of 10 bytes in the database file, although each memo field entry may contain up to 64K. 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, including 10 bytes for each memo field. 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.

CREATE/MODIFY STRUCTURE

Tips

MODIFY STRUCTURE makes backup copies of the database file (.bak extension) and memo file (.tbk extension) in the same directory as the original file. After the structure modifications are completed, the contents from the backup files are appended into the modified database file. Since MODIFY STRUCTURE is not able to create backup files if the disk or current directory is full, make sure that you have enough space available for the backup files before you modify a file's structure.

WARNING
The .bak file created with each MODIFY STRUCTURE is overwritten with the next MODIFY STRUCTURE.

Because MODIFY STRUCTURE appends data into the new file, you may lose 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.

Do 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, you change field positions and could lose data. You can, however, change field widths or data types at 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 been converted correctly.

If you convert numeric fields to character fields, dBASE IV will convert numbers from the numeric fields to right-justified 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 IV 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.

To convert incompatible data types (such as logical to numeric), first add a new field to the file, use REPLACE to convert the data, then delete the old field.

To use a database file as a SQL table in SQL mode, you must define the file as a table using the SQL DBDEFINE command. The DBDEFINE command updates the system catalogs that SQL uses to access the file.

However, you can use most dBASE commands to access database files in SQL mode (although you cannot use MODIFY STRUCTURE to change a database file that has been defined as a SQL table).

See Also

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

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

You must provide a filename for the view file, or use the ? (query clause) in the command line. 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.

Options

If a catalog is open, the ? symbol queries the catalog for all .vue files. If a catalog is not open, the ? symbol presents all .vue files on the disk. You can then choose a file to overwrite.

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

CREATE/MODIFY QUERY/VIEW, SELECT, SET FIELDS, SET FORMAT, SET INDEX, SET RELATION, SET VIEW, USE

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 the command is executed from a called procedure, any command in the procedure following DEACTIVATE is not executed (it does an immediate RETURN). If an ON PAD is in effect, DEACTIVATE MENU 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 or program.

DEACTIVATE POPUP returns control to the program line immediately following the one that activated the popup. If the ON SELECTION command is used to call a procedure that DEACTIVATEs the popup, any command in the procedure following the DEACTIVATE command is not executed. DEACTIVATE POPUP returns control to the line immediately following the ACTIVATE POPUP command in the calling procedure.

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 POPUP, RELEASE POPUPS, RESTORE SCREEN, SAVE SCREEN

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, compiles and executes a program or procedure, but also calls the dBASE IV full-screen debugger.

The debugger screen contains four windows that provide information about the state of the .prg file and open files. The windows 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. Press **Esc** or **Ctrl-End** to return to the **ACTION:** prompt from the breakpoint window.

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. If you exit from a DEBUG session, use the RESUME command to continue the suspended session. If you enter a second DEBUG command while the current session is suspended, you will execute a RESUME and return to the current DEBUG session.
- ↵ — 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, LINENO(), MODIFY COMMAND, PROGRAM(), 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. An array can have the same name as a dBASE IV command; however, this may cause unpredictable results.

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. Rows in an array correspond to records in a database, and columns in an array correspond to fields in a database. 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. The maximum size of an array (rows \x columns) is 1170.

```
. 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, as does the following:

```
. mdate[2,2]={6/15/88}
```

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, APPEND and REPLACE, have special forms (COPY TO ARRAY, APPEND FROM ARRAY, and REPLACE 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 FROM ARRAY, AVERAGE, CALCULATE, CLEAR ALL, CLEAR MEMORY, COPY TO ARRAY, COUNT, LIST/DISPLAY MEMORY, PUBLIC, RELEASE, REPLACE FROM ARRAY, RESTORE, SAVE, SUM

Getting Started provides information on memory allocations.

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 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.

If SET STATUS is ON, the message 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. If SET STATUS is OFF, message location is determined by the AT clause of the SET MESSAGE 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 prevent the user from selecting the separator line. A logical memory variable like Medit in this example can be defined to make the Delete option available only while Medit evaluates to a logical false (.F.). 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(), SET MESSAGE, SET STATUS, 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 printed report text for enhanced appearance.

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 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.). The box will print when _pline equals the top row of the box; the box will be part of the streaming output of dBASE.

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:

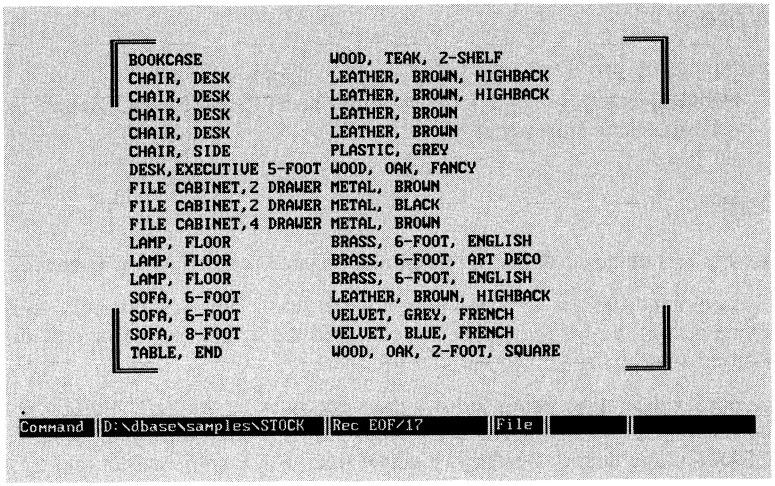


Figure 2-2 Boxed output

See Also

@...TO, SET BORDER, _box

Chapter 1, "Essentials," describes how to control printer output with system memory variables.

DEFINE MENU

Use DEFINE MENU in conjunction with the DEFINE PAD command to define a 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.

If SET STATUS is ON, the optional message field appears centered at the bottom of the screen; otherwise, the location is determined by the AT clause of the SET MESSAGE command. 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

ACTIVATE MENU, DEACTIVATE MENU, DEFINE PAD, MENU(), ON PAD, ON SELECTION PAD, PAD(), SET MESSAGE, SET STATUS

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. Menu prompts can be positioned anywhere on the screen where there is room for the prompt message. 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 and incrementing the row coordinates for each pad, or by using a popup.

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 the first line (line zero) 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. If SET STATUS is ON, the message appears centered at the bottom of the screen when the cursor is on the pad associated with it; otherwise, the location is determined by the AT clause of the SET MESSAGE command. 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

ACTIVATE MENU, DEFINE MENU, ON PAD, ON SELECTION PAD, PAD(), SET MESSAGE, SET SCOREBOARD, SET STATUS

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. 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.

To navigate a popup, use the ↑ and ↓ keys, or the first letter of the prompt. To select a file or a prompt action, press ←┘.

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 unless you reposition it with the AT clause of the SET MESSAGE command. If SET STATUS is ON, you cannot reposition the message; it is always centered at the bottom of the screen. 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 of the menu Main in the example of the DEFINE PAD command.

See Also

ACTIVATE POPUP, BAR(), DEFINE BAR, ON SELECTION POPUP, POPUP(), PROMPT(), SET MESSAGE, SET STATUS

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 **Shift-PrtSc** 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, SET STATUS

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. If SET STATUS is OFF and SET SCOREBOARD is ON, the **Del** marker is displayed on the scoreboard.

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, SET SCOREBOARD, SET STATUS, ZAP

DELETE FILE

DELETE FILE is an alternate form of the ERASE command. See the ERASE command entry in this chapter.

DELETE TAG

DELETE TAG deletes the indicated tags from a multiple index (.mdx) file if tag names are specified.

Syntax

DELETE TAG <tag name 1> [OF <.mdx filename>]
[,<tag name 2> [OF <.mdx filename>]...]

Usage

Multiple index (.mdx) files may contain up to 47 tags, each of which may impose 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 tags in any active .mdx file, you can remove it with the DELETE TAG command. Deleting a tag permanently removes it from the .mdx file, and restores space to the file. DELETE TAG opens a slot for a new tag 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, 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 also updated.

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.

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 Clcnt OF Client
```

See Also

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

DIR

DIR displays a list of the database files in the current directory.

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 DIR command differs from the DOS DIR command. In dBASE IV, if you type DIR followed by the name of a subdirectory of the current directory, the message **None** appears. This is because dBASE IV was looking for a database file by that name. The same command entered on the DOS command line would display a list of the files in the named directory.

In the dBASE DIR command, the path specification requires a terminating backslash. In dBASE IV, if you type:

```
. DIR \DBASE\SAMPLES
```

the message **None** appears. Entered at the DOS prompt, the same command would list the files in the Samples directory. To obtain the same listing at the dot prompt, you would have to enter:

```
. DIR \DBASE\SAMPLES\
```

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 DEFAULT, SET DIRECTORY

DISPLAY

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 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

NOTE
All dBASE IV 1.0 object code files must be recompiled in dBASE IV 1.1 in order to run under version 1.1. Recompilation will in most instances occur automatically. The commands that run a dBASE object file will check the version of the code file and recompile it if it is a version 1.0 code file, and the corresponding source file can be found. dBASE IV will search the current directory and all directories in the dBASE PATH for source files. The recompile will fail if the source file has been renamed or cannot be found, or if the code file was created with the DBLINK utility.

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. After locating the first instance, DO does not look for other procedures with the same name.
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 .dbo file extension after compilation. You also should not rename a .dbo file. DOing a renamed .dbo causes a **Procedure not found** error message when dBASE IV is unable to locate the main procedure new name of the .dbo.

The dBASE IV internal text editor, which can be accessed from MODIFY COMMAND or the Control Center, will delete an old .dbo file when a .prg file is modified. A subsequent DO command will recompile the .prg file and create a new .dbo file before executing the procedure. As other text editors do not delete the old .dbo file, SET DEVELOPMENT allows you to verify that DO does not execute an outdated .dbo file.

If SET DEVELOPMENT is ON and you use an external text editor to edit existing program source (.prg) files, DO compares the time and date stamp of the 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.

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 50 parameters. No more than ten of the parameters can be literal values, and no more than six can be array elements; the rest must be variables. 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. The error message **Procedure/function call nested too deep** may eventually result. dBASE IV allows a default of up to 16 DOs, but the number of nested DOs is limited by the amount of memory you have.

Use a RETURN rather than a DO to return control to the calling program. 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 compiled file (in this case, .fro).

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.

DO CASE

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.

dBASE IV limits the number of nested DO WHILEs to 32 per procedure.

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

&(macro), 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** or **F9 Append** 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. EDIT reverts to full screen, and cannot be used in a window even if you call it from a BROWSE window by pressing **F2 Data**.

When the EDIT command is completed, you return to your point of origin: the dot prompt, the next line of a .prg file, or the Control Center.

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 if you move the cursor to the end of the file, and you can mark records for deletion.

NODELETE prevents you from deleting records during the edit.

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.

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.

Navigation Keys

The following table lists the navigation keys that you can use in EDIT:

Table 2-6 Navigation Keys

Key	Action
F1	Help
F2	BROWSE
F3	End of record
F4	Beginning of record
F5 - F9	No action
F10	Records menu
Shift-F2	Transfer to query design
Shift-F3 - Shift-F7	No action
Shift-F8	Copy same field from previous record
Shift-F9	Print menu
Shift-F10	Macro (keyboard) processing

See Also

BROWSE, CHANGE, CREATE/MODIFY QUERY/VIEW, MODIFY COMMAND, SET DESIGN, SET FIELDS, SET FORMAT, SET LOCK, SET REFRESH, SET WINDOW OF MEMO

EJECT

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 PRINTER 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 and __pcolno system memory variables 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.

EJECT PAGE

See Also

?/??, EJECT, ON PAGE, SET ALTERNATE, SET PRINTER, _padvance, _pageno, _plength, _plineno

Chapter 1, "Essentials," discusses system memory variables and streaming output.

ERASE/DELETE FILE

ERASE removes a file from the disk directory. (DELETE FILE is an alternate form of ERASE.)

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

WARNING
Even with SET SAFETY ON, ERASE does not ask for confirmation.

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 if an .mdx file exists.

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(), SET SAFETY, 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 or ORDER 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 considerably 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, SET RELATION, TAG(), USE

FUNCTION

The FUNCTION command defines a user-defined function (UDF).

Syntax

```
FUNCTION <UDF name>
    [PARAMETERS]
    .
    .<other dBASE commands>
    .
RETURN <expression>
```

Usage

A user-defined function is a special type of procedure that can be used in a dBASE expression and returns a value. Its definition consists of the FUNCTION command followed by the name of the procedure you associate with this definition. A UDF name can be up to eight characters long, must begin with a letter, and cannot contain spaces.

The UDF name may be followed by an optional parameter list, dBASE commands or program structures, and must end with a RETURN command which specifies the value that the UDF returns. Depending on program complexity, you may nest up to a maximum of 32 other UDFs within the calling UDF. In practice, this is almost always a smaller number.

The optional PARAMETERS command assigns names to input parameters used as arguments by the UDF. If you want to pass parameters to a UDF, the parameters command is no longer optional: it must be the first command in the user-defined function definition. These parameter names are local to the UDF. The parameters must be passed to the UDF in the same order in which they appear in the parameters list.

The RETURN command returns the <expression> specified in the function definition.

When dBASE IV encounters a UDF name in a command line, it searches for and executes the UDF and returns a single value. When a UDF terminates, program control returns to the calling command.

WARNING
Do not use the name of an existing dBASE function for a UDF. dBASE IV will execute its own function and ignore any UDF that has the same name.

Interrupt Handling

A dBASE IV command is interrupted when its execution is temporarily halted in order to execute another dBASE IV command, a UDF, or a program. Most dBASE programs can be interrupted by a user-defined function, or an ON command at multiple program levels.

The interrupting UDF is not allowed to close or change the structure of the active database file or clear or release the active window (if any). For example, while in BROWSE, if you call a UDF that tries to USE a different database in the same work area as the BROWSE, you will receive an error message. This is because dBASE IV preserves the work area and any active window so that you can return to them.

dBASE IV permits certain changes without restrictions and propagates these changes up to the interrupted command. For example, an ON KEY command can call a UDF which changes a value in a BROWSE field. The changed value is passed to BROWSE and the field is altered.

In order to preserve the original database file, when DBTRAP is set to ON, any commands that would close or change the structure of the original database file are blocked.

The commands that are blocked include USE, CLOSE DATA, CONVERT, PACK, MODIFY STRUCTURE, and ZAP.

In order to preserve the original window, users are not allowed to remove it from memory. Commands that are blocked include RELEASE WINDOWS, CLEAR WINDOW, DEFINE WINDOW <same window name>, and RESTORE WINDOW <same window name>.

You may MOVE a window or change its colors by issuing the SET COLOR command inside the window.

When the interrupt is over, the window that was active is restored. If you have moved or changed the color attributes of the window inside the interrupting UDF, when you return, the window will be in the new location and display the new colors.

Interruptions may occur for many reasons. Listed below, for example, are many ways that a BROWSE command can be interrupted with a UDF:

- LOCK <expN>
- WIDTH <expN>
- SET FIELDS <calculated field name> = <exp>
- SET RELATION TO <exp>
- SET FILTER TO <expL>
- @...RANGE <low exp>, <high exp>
- @...VALID <expL>

- @...WHEN <expL>
- @...DEFAULT <exp>
- @...MESSAGE <expC>

The WIDTH expression is evaluated immediately. The @...RANGE and VALID expressions are evaluated only if you make a change to the field or if you include the REQUIRED keyword in the command syntax.

WARNING
Do not change the operating conditions of the interrupted command within a UDF. These include the relation chain, the field list, and the filter and scope conditions of the database in use at the time of the interrupt. If you change these, the results will be unpredictable when you return from the UDF.

Non-Recursive Commands

A group of closely-related commands cannot call each other in the same work area. This is true regardless of the setting of DBTRAP. These commands are:

BROWSE, EDIT, CHANGE, INSERT, and APPEND

Two other closely related commands cannot call each other even in separate work areas, regardless of the setting of DBTRAP. These commands are:

LIST and DISPLAY

In addition, Control Center design screens, ASSIST, and CREATE/MODIFY cannot be called recursively.

Indexing with UDFs

When DBTRAP is ON, the indexing process is protected from all interrupts. To create an index that uses a UDF in the key expression, you must set DBTRAP to OFF and use either syntax:

INDEX ON <UDF name>...[FOR <UDF name>]

Indexing with a UDF in the key expression can create an invalid index. The indexing procedure does not give any error messages if the UDF does not provide an indexable value.

If the indexing procedure executes correctly, but the UDF is not found by the index at a later date, dBASE IV will put a default value into the index and thus create an invalid index. You must be aware of both potential problems with a UDF-containing index key.

SQL Exceptions

A UDF definition can be placed in a .prs file, and a SQL program may call a UDF with a dBASE command that is used by the .prs file.

SQL language syntax places some limits on UDF usage:

- You cannot use UDFs in a SQL statement
- You cannot use SQL statements in a UDF
- You cannot use dBASE commands that are prohibited in SQL mode in a UDF that is in a .prs file.

See Appendix H for the list of dBASE commands permitted in SQL mode.

& Macro Substitution

& macro substitution is allowed in UDFs.

Examples

The following UDF, Plus_tax, returns the final cost of an item by adding the tax to the price. This UDF 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 DBTRAP, SET PROCEDURE

Chapter 1, "Essentials"

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> [IN <alias>]

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 (enclosed in parentheses if it is a memory variable)

- A character expression that yields a letter A through J or an alias name

See the Filenames section in Chapter 1 for more information on the use of filenames and aliases.

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(), SELECT, 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 on the RUN command with:

```
. HELP RUN
```

See Also

SET HELP

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 a dBASE II file, change the file extension to .db2 before importing the 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 the output file on the same drive and directory as the original file and puts it into USE. Then it writes the imported data to the new output file. To display information about the imported file, use DISPLAY STATUS and DISPLAY STRUCTURE.

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, DISPLAY STATUS, DISPLAY STRUCTURE, 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>]
        [FOR <condition>] [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. If you do not use TAG, an .ndx file is created.

Without a FOR clause or the UNIQUE option, the index will contain all records. The FOR clause can be used only with .mdx tags. UNIQUE can be used with .ndx files and .mdx tags.

INDEXing occurs in ascending order, unless you use the DESCENDING option. You may use DESCENDING only when building .mdx tags.

Usage

Indexed database files allow you to move 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. The controlling index controls the movement of the record pointer in the database file.

The index, which is written to disk as either an index file or as a tag in a multiple index file, contains the key values and the corresponding record number for each record in the database file. The physical order of the records in the original database file on disk is not changed by the INDEX command.

A multiple index file may contain up to 47 tags, each of which can impose an index order on the database file. Tag names follow the same rules as variable names: they may be up to 10 characters long, must begin with a letter, and may contain letters, numbers, and underscores. Index filenames (both .ndx and .mdx) follow the rules for all filenames: they may be up to eight characters long, and may only contain characters allowed by the operating system.

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>.

Once you create an index, it becomes the new controlling index, and records appear in the new index order. To change the controlling index, use the SET INDEX or SET ORDER command. Additional active indexes have no effect on record pointer movement, and are open only so they can be updated when data in their keys is changed in the database file. Whenever changes are made 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 key expression determines whether records will be ordered chronologically (date expressions), numerically (numeric expressions), or in ASCII order (character expressions). When the 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

Using FOR <condition> indexes only the records that meet a specified condition. FOR can be used only with .mdx tags, not with .ndx files. If no records meet the FOR condition, an empty tag will still be created and maintained.

The FOR condition cannot include calculated fields. The maximum length of the condition is 220 characters (including embedded spaces).

Using 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 encountered with that value is included in the index. 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.

dBASE IV processes UNIQUE indexes only once. Therefore, a previously hidden key value is not automatically updated when it is changed. REINDEX explicitly updates all key values in a UNIQUE index.

DESCENDING builds an .mdx tag in descending numeric 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. You can create 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().

Operations that move sequentially through the database file are usually slower if an index is open. Use the SET ORDER command if you are not using the index to position the record pointer, but still want to update the index keys. SET ORDER speeds up data access if an .ndx file is open.

LIST/DISPLAY STATUS will show the FOR <condition> for indexes (if a condition has been specified) after the line showing the <key expression>.

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 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. 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.

INDEX ignores filters set with SET FILTER or SET DELETED. All records of the database file are included in the index unless you include FOR <condition> or use the UNIQUE option to suppress duplicate keys.

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. Opening an index that references a non-existent memory variable causes an error.

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 creating an .mdx tag with the INDEX command. If the file is not USEd EXCLUSIVEly, the error message **Exclusive use of database is required** appears.

You cannot use the INDEX command in an active transaction if:

- a non-production index file is currently open
- it overwrites an existing .ndx or .mdx file
- it overwrites an existing .mdx tag

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 amount of their transactions:

```
. INDEX ON Client_id + STR(Total_bill,10,2) TO By_amnt
  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
     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 a conditional index containing only records with a total bill of $1000 or more:

```
. INDEX ON Client_id TAG Large FOR Total_bill >= 1000
  100% Indexed              6 Records Indexed
. LIST

Record#  CLIENT_ID  ORDER_ID  DATE_TRANS  INVOICED  TOTAL_BILL
      1  A10025     87-105    02/03/87    .T.         1850.00
     12  A10025     87-116    04/10/87    .F.         1500.00
      2  C00001     87-106    02/10/87    .T.         1200.00
      4  C00001     87-108    02/23/87    .T.         1250.00
      3  C00002     87-107    02/12/87    .T.         1250.00
      7  L00002     87-111    03/11/87    .F.         1000.00
```

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

BEGIN/END TRANSACTION, CLOSE, COPY INDEX, COPY TAG, CTOD(), DAY(), DELETE TAG, DTOC(), DTOS(), FIND, KEY(), LIST STATUS, LOOKUP, MDX(), MONTH(), NDX(), ORDER(), REINDEX, SEEK, SEEK(), SET DELETED, SET EXCLUSIVE, SET FILTER, SET INDEX, SET NEAR, SET ORDER, SET SAFETY, SET UNIQUE, SORT, STR(), SUBSTR(), TAG(), USE, VAL(), YEAR()

The SORT command explains the differences between INDEX and SORT.

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.

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. 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. Pressing **PgDn** or **PgUp** copies the preceding record entirely into the newly-inserted blank record.

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, or 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.

Tip

To copy the contents of the preceding record to the INSERTed record, use SET CARRY ON before INSERTing records. SET CARRY TO also lets you copy specific fields.

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, BROWSE, CHANGE, EDIT, MODIFY COMMAND, READ, REPLACE, SET CARRY, SET EXCLUSIVE, 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. 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.

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**.

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.

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!

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

KEYBOARD

KEYBOARD enters a series of characters into the type-ahead buffer. dBASE IV then reads the characters as if the user typed them in.

Syntax

KEYBOARD <expC> [CLEAR]

Usage

dBASE IV does not read the characters in the type-ahead buffer until you execute a command that expects keyboard entry. To supply keystrokes to commands such as BROWSE, EDIT, and READ, first enter the keystrokes with the KEYBOARD command, then execute the command that uses the keystrokes.

KEYBOARD respects the size of the type-ahead buffer, truncating any excess characters. The type-ahead buffer has a default capacity of 20 characters, which can be changed with SET TYPEAHEAD.

You can insert all ASCII characters that are returned by the CHR() function for values 1-255. CHR(0) is ignored if input.

You can place control keys such as ↵, **Home**, and **Esc** into the type-ahead buffer with the CHR() function. The values for these keys are the same ones returned by the INKEY() function (see INKEY() in Chapter 4). INKEY() values for some keys (such as the function keys and **Alt** key combinations) are negative and, therefore, cannot be used with KEYBOARD and CHR(). However, you can use a macro to record and playback these keys.

Options

If there are any unread characters in the type-ahead buffer, the CLEAR option clears them before inserting the new characters.

If the CLEAR option is not used, the buffer is not cleared automatically.

Special Cases

You may enter ASCII special characters using the **Alt** key and the numeric keypad.

If you use the KEYBOARD command to enter a null string or a string of length 0, the command will execute and return without errors, but nothing will be entered into the type-ahead buffer.

Example

The following program segment displays a BROWSE screen using the NOCLEAR option. The KEYBOARD command inserts **Esc** into the type-ahead buffer, which deactivates BROWSE, leaving the table displayed.

```
USE Stock
KEYBOARD CHR(27)   && Escape from the BROWSE Screen
BROWSE NOCLEAR
```

See Also

CHR(), CLEAR TYPEAHEAD, INKEY(), PLAY MACRO, READKEY(), RESTORE MACROS, SAVE MACROS, SET TYPEAHEAD

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 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.

If SET ECHO is ON, source code from the .lbg file is printed instead of the labels.

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.

Special Cases

In multi-user operations, the LABEL FORM command places an automatic lock on the database file before printing if SET LOCK is ON. If another user has used RLOCK() or FLOCK() on the file, LABEL FORM generates the **File is in use by another** error message. You may copy the file to a temporary file to generate the report. LABEL FORM does not lock the file if SET LOCK is OFF.

See Also

CREATE/MODIFY LABEL, FLOCK(), RLOCK(), SET LOCK, 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 mixed uppercase and lowercase 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.

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 DEFAULT, SET DIRECTORY

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.

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 display for numeric variables contains both the number and the type (type F or type N).

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.

LIST/DISPLAY MEMORY

For every dBASE IV 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 IV 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 that you create, the **User MEMVAR/RTSYM Memory Usage** section displays the memory allocation totals.

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

ACCEPT, AVERAGE, CALCULATE, COUNT, DECLARE, DEFINE MENU, DEFINE PAD, DEFINE POPUP, DEFINE WINDOW, INPUT, PRIVATE, PUBLIC, RELEASE, RESTORE, SAVE, STORE, SUM

Chapter 5, "System Memory Variables," includes a discussion of the system variables displayed by this command.

Getting Started 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, DESCENDING, or has a FOR condition
- Open memo filenames
- Filter formulas
- Database relations
- Format files

In addition, it shows:

- File search path
- Default disk drive
- OS working drive/directory
- 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). The file header contributes to the total number of bytes in a record.

If the number of fields in the database exceeds a screenful (where the active screen can be a small window or a full display of 43 lines), then DISPLAY STRUCTURE pauses after a screenful 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 FROM, CREATE/MODIFY STRUCTURE, 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 DBNETCTL.300 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 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.

Options

<binary filename> can be either a filename or a character expression that results in a filename.

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.

Binary files created for dBASE III PLUS can be used with dBASE IV if they use a single null-terminated character string parameter. Binary files that depend on the way dBASE III PLUS arranges memory variables in memory will not work with dBASE IV.

While designing the assembly language program from which the binary file is made, you must conform to the following specifications:

- The first executable instruction must originate (ORG) 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 (RETF). Most commercial programs end with an exit call rather than a far return; execute these with the RUN/! command, not with LOAD and CALL.

You can pass up to seven parameters to the binary file with the CALL command or the CALL() function. The parameters may be field names, expressions, memory variables, or array elements of any data type except memo.

Before dBASE IV passes control to the binary program, it evaluates each parameter and converts the result to a null-terminated character string.

The CX register contains the number of parameters passed. DS:BX contains the address of the first parameter. ES:DI points to a 28-byte block of memory that is a list of seven four-byte addresses. The first four bytes point to the first parameter, the next four bytes point to the second parameter, and so on. If fewer than seven parameters are passed, the remaining addresses point to null-terminated empty strings.

The binary program can alter the contents of any of the parameters. However, only changes made to parameters passed as memory variables are retained when the program returns. dBASE IV converts the character string representation of each memory variable back to the original data type, retaining the orginal data length.

To prepare an assembly language program written in 8086/8088 assembler for LOADing by dBASE IV, use an assembler program to assemble the source file (.asm) into an object file (.obj), link the object file into an executable file (.exe), and then create a binary file from the executable file. Following are the commands you would use with the Microsoft Macro Assembler (MASM).

MASM <source>;

LINK <target>;

EXE2BIN <target>

Examples

Following is an assembly program that substitutes all occurrences of a specified character with another specified character. This program is also included on your Samples disk, along with the binary file Strsubst.bin.

```
; Strsubst.asm (Source for Strsubst.bin)
;
; Substitute characters in a character string.
;
CODE    SEGMENT BYTE PUBLIC 'CODE'
STRSUB  PROC    FAR
ASSUME  CS:CODE
        PARAM1  EQU   ES:[DI+0]
        PARAM2  EQU   ES:[DI+4]
        PARAM3  EQU   ES:[DI+8]
        PARAM4  EQU   ES:[DI+12]
        PARAM5  EQU   ES:[DI+16]
        PARAM6  EQU   ES:[DI+20]
        PARAM7  EQU   ES:[DI+24]
START:
    PUSH  BP                ; Save stack frame
    MOV   BP, SP
; Quit if there aren't at least 3 parameters.
    CMP   CX, 3
    JL    DONE
; Load first byte of 2nd parameter in CL
    LDS   BX, PARAM2        ; DS:BX points to 2nd parameter
    MOV   CL, [BX]          ; Store 1st byte
; Load first byte of 3nd parameter in CH
    LDS   BX, PARAM3        ; DS:BX points to 3rd parameter
    MOV   CH, [BX]          ; Store 1st byte in CH
; Point DS:BX to 1st parameter
    LDS   BX, PARAM1
; Loop for each character in 1st parameter
AGAIN:  MOV   AL, [BX]          ; Get next character in AL
        CMP   AL, 0             ; Is it end of string?
        JE    DONE              ;    Yes: exit
        CMP   AL, CL            ; Is it character we're searching for?
        JNE   NEXT              ;    No:  don't replace
        MOV   [BX], CH          ;    Yes: replace
NEXT:   INC   BX                ; Point to next
        JMP   AGAIN             ;
DONE:   POP   BP                ; Restore stack frame
    RET
STRSUB  ENDP
CODE    ENDS
        END
```

This example executes the binary file Strsubst.bin.

```
. LOAD Strsubst
. Mstr1 = "* * * * * *"
. CALL Strsubst WITH Mstr1, "*", "#"
. ? Mstr1
# # # # # #
```

See Also

CALL, CALL(), LIST/DISPLAY STATUS, RELEASE, RUN, 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 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 than the LOCATE command because SEEK follows the index order. 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 ORDER TO ↵, LOCATE, and SET ORDER to <expN> 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, 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, QUIT

MODIFY COMMAND/FILE

MODIFY COMMAND/FILE calls the dBASE IV full-screen text editor or a specified word processor.

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

By default, MODIFY COMMAND/FILE calls up dBASE IV's full-screen text editor to create or edit program files, format files, or any standard ASCII text 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 a window is referenced which has not yet been activated with the ACTIVATE WINDOW command or has been closed with the CLEAR WINDOWS command, 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 internal text editor by setting TEDIT in the Config.db file. See *Getting Started* for information about Config.db settings.

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 that you edit a file, a backup copy of the previous version is saved in the same directory as the original file. Whether or not SET SAFETY is ON, any earlier version of a backup file is automatically overwritten each time its original file is changed.

The name of a backup file is the same as that of the original file. The extension of the backup file varies according to the extension of the original file, as shown in the following table:

Table 2-7 Backup file extensions

Original file extension	Backup file extension
.prg	.bak
.txt	.bak
.dbf	.dbk
.dbt	.tbk
.mdx	.mbk

WARNING
Do not use one of the backup extensions for an original file. If you do, after edit dBASE IV will not make a backup copy or warn you that your original file is being overwritten.

After you edit a .prg file, the editor deletes the old .dbo file. You must compile the new .prg file to create a new .dbo file.

The MODIFY COMMAND screen contains a bar menu of editing commands. A complete list of the cursor movement keys 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 DEVELOPMENT, TYPE

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, 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. You can use any of the three forms of indicating a note anywhere in a program. If a NOTE or && line ends with a semicolon, dBASE IV reads the next line as part of the comment line.

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 syntax errors and evaluation errors (among others). 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. ON KEY does not work if SET TYPEAHEAD is 0.

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.

The following full-screen commands trap and handle errors internally. Therefore, the <command> associated with ON ERROR will not be triggered if an error occurs inside of one of these commands:

APPEND
ASSIST
BROWSE
EDIT
CREATE/MODIFY APPLICATION/LABEL/REPORT/SCREEN/
    STRUCTURE/VIEW
MODIFY COMMAND/FILE

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). Use RETURN in an error procedure to continue with the next line after the error. If the next line is a DO WHILE/IF loop, RETURN continues in the loop.

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.

You should avoid using a dBASE command recursively with the ON KEY command. See the FUNCTION command for the list of commands that should not be used recursively. The same limitations on recursive commands apply to UDFs, ON KEY, and ON ESCAPE.

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 uppercase or lowercase characters.

The <key label name> for printable keys are the characters, digits, or symbols you see. Table 2-8 shows you the <key label name> descriptions to use for the non-printing keys of your keyboard.

A READ takes precedence over ON ESCAPE. If a READ is active and you press the **Esc** key, the next line of code after the the READ is executed. To trap the **Esc** key (27) after a READ, use the LASTKEY() or READKEY() function.

Table 2-8 <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

(continued)

Table 2-8 <key label name> descriptions for non-printing keys (*continued*)

Keycap Identification	<key label name>
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 . . .

* On most keyboards, ON KEY LABEL **Ctrl-M** can be used to trap the ↵ key.

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
* error processing commands
.
.
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, SET TYPEAHEAD, 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.

User-defined menu commands should be issued in the following order:

1. DEFINE MENU
2. DEFINE PAD <pad name> OF <menu name>
3. ON PAD <pad name> ACTIVATE POPUP
4. DEFINE POPUP <popup name>
5. DEFINE BAR <line number> OF <popup name>

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 passes a particular line on the current page during a ?/?? or EJECT PAGE command. Typically, ON PAGE is used for 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 the 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 do not 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 LIST.

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. You should end procedures that involve report bands with a ?? because ? will issue a carriage return before the next report band, leaving a blank line at the top of the page.

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 IV 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

?/??, FUNCTION, 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 IV doesn't compile external program modules while running a program.

See Also

@, APPEND, BROWSE, CHANGE, EDIT, INSERT, ON ERROR, READ, SET FORMAT

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 descendant 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 selection will not do anything when chosen from the pop-up menu. You could use this feature to disable a selection for specific users. If you use ALL for the pop-up name, the command you indicate applies to all of the popups.

User-defined menu commands should be issued in the following order:

1. DEFINE POPUP.
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.

NOTE
After using the ON SELECTION POPUP command, the active pop-up menu is redrawn when it is reactivated. However, the screen is not saved before executing any of the ON commands. You must save and restore the screen as required in your program if any of the ON commands are executed.

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.

Special Cases

In a network environment, the database file must be in exclusive use before you issue the PACK command.

See Also

COPY, DELETE, DELETED(), DIR, RECALL, REINDEX, SET AUTOSAVE, SET EXCLUSIVE, ZAP

PARAMETERS

PARAMETERS assigns local variable names to data items passed from a calling program. This command receives variables passed by the DO or DEBUG command, or by a user-defined function.

Syntax

PARAMETERS <parameter list>

Usage

In a program file, PARAMETERS must either be the first executable command, or must immediately follow a PROCEDURE or FUNCTION 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, but no more than ten can be literal values and no more than six can be array elements; the rest must be variables.

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.

If you pass an array element as a parameter to a subroutine (the element is passed by value, not by reference), the change is local to that routine. The original value remains when you RETURN from the calling routine. To change the value of an array element, use the array name as a memory variable instead of passing it as a parameter.

See Also

@, DO, FUNCTION PROCEDURE, 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

KEYBOARD, 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"
- Print the contents of the print buffer the number of times defined by _pcopies. If _pcopies is greater than 1, dBASE IV spools the output to the hard disk, then prints it the number of times specified 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. Also note that you can use the system memory variables _pbpage and _pepage to set the beginning and ending page range numbers for printing with PRINTJOB.

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 private system memory variable in a dBASE IV procedure, a PRIVATE copy is created and it is assigned the current value for that system variable. When you leave the procedure, the original system memory variable's value is re-evaluated and reset.

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 for the PROCEDURE filename, and locate the first instance of that PROCEDURE name.
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 963 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 nine characters long. They may contain letters, numbers, and underscores, and may begin with a letter or a number. 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 Proc1.prg

PROCEDURE Message1
   @ 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: Proc1.prg
```

PROCEDURE

To activate a procedure within Proc1.prg:

```
. SET PROCEDURE TO Proc1
. DO Message1
```

See Also

COMPILE, DEBUG, DO, FUNCTION, 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 2-9 summarizes the database security types, how to implement each security type, and the results of security implementation.

Table 2-9 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 2-9.

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:

```
. COPY FILE Filename.crp TO Filename.dbf
```

2. Delete the .crp file, as follows:

```
. ERASE Filename.crp
```

3. Rename the .cpt file so that it has a .dbt 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, "Using Tools," in *Using the Menu System* 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 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 PUBLIC command syntax.

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.

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 declaring a parameter of the same name in a procedure.

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.

Examples

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
```

Within a dBASE program file, the following command creates two public arrays, named Parts and Items:

```
PUBLIC ARRAY Parts [6,3], Items [8, 3]
```

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.

If a format file is open, READ activates it after it puts the @...GETs on the screen and overwrites the screen. Be sure to close format files after editing to avoid READing format files and overwriting the screen; otherwise, make sure that all @...GETs are included in the format file. The format takes precedence in controlling the screen.

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, ON ESCAPE, 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.

If SET DELETED is ON, you must move the record pointer to the record to be RECALLed with GOTO RECORD <n>, or you can specify the record number with RECALL RECORD <n>. When SET DELETED is ON, RECALL ALL will not recall any records.

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 can REINDEX .mdx files.

Example

```
. USE Transact
. REINDEX

Rebuilding index - C:\DBASE™RANSACT.MDX Tag: CLIENT_ID
   100 % indexed          12 Records indexed
Rebuilding index - C:\DBASE™RANSACT.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, screens, 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>]
/SCREENS [<screen 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. In a program, RELEASE ALL does not delete public variables.

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. When specifying the module name, do not include the .bin extension.

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 SCREENS releases saved screen images from memory. The screen name list contains the names of previously saved screens. If no screens are listed, RELEASE SCREENS releases all saved screens.

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, DEFINE MEMO, DEFINE POPUP, DEFINE WINDOW, LOAD, ON PAD, ON SELECTION PAD, ON SELECTION POPUP, RESTORE, RETURN, SAVE, SAVE SCREEN, 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 cannot RENAME an open file, and the new filename cannot be an existing filename in the same directory.

If you rename a database file that has associated files (such as memo or index files), you must also rename the corresponding files to match the new database filename. It is preferable to rename a database file (and its associated files) by accessing it through MODIFY STRUCTURE and saving it with a new name. However, this does not rename the production .mdx file.

You cannot rename a production .mdx file. The name of a production .mdx file is internal to the file and RENAME cannot change it. One way to change the name of a database file and its associated .mdx file is to use the COPY STRUCTURE EXTENDED command to copy the database structure to a new file, and then use the CREATE FROM command to create a new database file from the structure extended file.

In general, it is not a good idea to rename database files because of possible CATALOG and non-production .mdx file conflicts.

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.

Example

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, MODIFY STRUCTURE, 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. When REPLACEing data in memo fields, the data type must be character.

The <scope> option is used to specify the NEXT[N], that is, the number of fields to replace.

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 the index is active.. If you specify a scope, FOR, or WHILE when making replacements on an indexed field, if the record is active 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. To REPLACE all the records, you should put the database file in natural order by entering SET ORDER TO ↵.

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

REPLACE FROM ARRAY

REPLACE FROM ARRAY replaces fields in a database file with data from an array.

Syntax

REPLACE FROM ARRAY <array name> [<scope>]
[FIELDS <field list>] [FOR <condition>]
[WHILE <condition>]

Defaults

The default scope is the current record in the active database file.

Usage

Data is copied from the first row of the specified array into fields of the active database file. The first array element is copied to the first field, the second element to the second field, and so on. The process stops when there are no more fields to replace, or there are no more elements in the first row of the array.

Use a FIELDS clause to specify fields to be replaced. The correspondence between array elements and fields in the database is determined by the order of fields in the fields list.

If you specify a scope, or use FOR or WHILE to select more than one record to replace, dBASE IV will replace fields in the selected records until all of the array elements have been exhausted.

The data types of the array elements must match the data types of the fields they will replace. dBASE IV displays **Data type mismatch** if data types of array elements and fields do not match. However, changes to fields already processed are not backed out. Memo fields cannot be replaced, since they cannot be stored in array elements.

If the database file (or field list) has more fields than the array has columns, values in the excess fields are not changed. If the array has more columns than the database file (or field list) has fields, the excess array elements are ignored.

REPLACE FROM ARRAY will respect any active SET FIELDS list, replacing values only in the database fields specified in the SET FIELDS list.

Example

In this example, the data in a record is copied to an array, edited in memory, and then replaced in the record.

```
USE Stock ORDER Order_id
DECLARE mstock[1,6]
morder = SPACE(6)
CLEAR
@ 10, 10 SAY "Order to update? " GET morder ;
         PICTURE "99-999"
READ
IF SEEK(morder)
   COPY TO ARRAY mstock NEXT 1
   CLEAR
   @  8, 13 SAY "Order ID:  " + mstock[1,1]
   @  9, 14 SAY "Part ID:"     GET mstock[1,2]
   @ 10, 12 SAY "Part name:"   GET mstock[1,3]
   @ 11, 10 SAY "Description:" GET mstock[1,4]
   @ 12, 12 SAY "Item cost:"   GET mstock[1,5] PICTURE "99999.99"
   @ 13, 13 SAY "Quantity:"    GET mstock[1,6] PICTURE "999"
   READ
   mok = .T.
   @ 15, 10 SAY "Save changes?" GET mok PICTURE "Y"
   READ
   IF mok
      REPLACE FROM ARRAY mstock
   ENDIF
ENDIF
```

See Also

APPEND FROM ARRAY, COPY TO ARRAY, DECLARE, PUBLIC

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. This command displays the reports design screen, which is described in Chapter 10 of *Using the Menu System*. 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.

If you want the settings from a particular print form file rather than the current environment's settings to be used, assign the name of that print form file to the __pform system memory variable.

Changes made to a .prf file are attached to the report only from the report design screen of the Control Center. Refer to Chapter 10 of *Using the Menu System*.

If SET ECHO is ON, the source code of the .frg file is printed instead of the report.

In multi-user operations, the REPORT FORM command places an automatic lock on the database file before printing if SET LOCK is ON. 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. REPORT FORM does not lock the file if SET LOCK is OFF.

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, INDEX, LABEL FORM, SET LOCK, SORT, _pform

Chapter 10 of *Using the Menu System*

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.

Special Cases

In a network environment, the database file must be in exclusive use before you issue the RESET command.

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 designate them PUBLIC before restoring them, and then use the ADDITIVE option. 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 *Getting Started* 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.

Memory variables are always restored as PRIVATE. To restore the memory variables as PUBLIC, first declare them as PUBLIC, then RESTORE the memory file with the ADDITIVE option.

See Also

DECLARE, PRIVATE, PUBLIC, RELEASE, SAVE, STORE

RESTORE MACROS

RESTORE MACROS restores macros that were saved to a 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 SCREENS

RESTORE SCREENS displays a previously saved screen image from its screen memory variable name.

Syntax

RESTORE SCREEN FROM <screen name>

Usage

The screen image must have been stored in memory with SAVE SCREEN TO <screen name> . Images can be restored multiple times from a single saved image.

Saved screens are removed from memory with RELEASE SCREENS or CLEAR ALL.

See Also

ACTIVATE SCREEN, CLEAR ALL, CLEAR SCREENS, RELEASE SCREEN, SAVE SCREEN

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

ACTIVATE WINDOW, 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 that caused an error.

Syntax

RETRY

Usage

Use RETRY primarily in error recovery procedures, since it can repeat a command until it is successfully executed. Note that RETRY will re-evaluate an IF or DO WHILE condition. This is in contrast to RETURN, which takes you to the next line after the error.

RETRY calls the command that caused the latest procedure to be run. The procedure that you execute with ON ERROR DO <procedure> should attempt to isolate the error and correct it. RETRY can then be used to return to the command line that caused the error and re-execute it. RETRY also clears out the value returned by the ERROR() function.

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."

You must include the RETURN command in a procedure or function because dBASE IV does not know what value to return for you.

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. RETURN has to include an <expression> if it is being compiled in a user-defined function.

A RETURN from an ON ERROR routine takes you to the line following the error.

See Also

COMPILE, DEBUG, 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 and then terminates the transaction. It closes and deletes all files created during the transaction.

Syntax

ROLLBACK [<database filename>]

Usage

Use this command to undo changes made to files after you have issued a BEGIN TRANSACTION command.

ROLLBACK compares the transaction log file with the active database files to undo the active transaction and restore the database files to what they were at the beginning of the transaction.

ROLLBACK cannot be used with a <database filename> during a transaction. It should be used only for recovering from system failures. Always try ROLLBACK without a <database filename> first. If ROLLBACK cannot process all the database files that were modified during the transaction, then you can use ROLLBACK<database filename> to roll back database files individually.

After a transaction has been rolled back, ROLLBACK removes the transaction log file and transfers control to the command after END 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

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
ON ERROR
```

See Also

BEGIN/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.

RUN executed in a window will pause after output until you press a key. This is to let you view command output before the screen is cleared.

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. ASSIGN will lock your computer without an error message. If you get a memory 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
```

See Also

CALL, CALL(), GETENV(), LOAD, RUN()

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

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

The alternate syntax for this command is:

SAVE ALL LIKE <skeleton> TO <expC>

Use SAVE TO to save memory variables that you may work with in subsequent dBASE IV sessions.

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

DECLARE, 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

KEYBOARD, PLAY MACRO, RESTORE MACROS

SAVE SCREEN

SAVE SCREEN lets you save an image of the current display to a memory variable name.

Syntax

SAVE SCREEN TO <screen name>

Usage

Use SAVE SCREEN in a program to save an image of the screen in memory. The image can then be redisplayed with the RESTORE SCREEN command. SAVE SCREEN is not related to the ACTIVATE SCREEN command.

If you use the same <screen name> as a previously saved screen, the original screen is replaced.

Saved screens are removed from memory with RELEASE SCREENS or CLEAR SCREENS.

The SAVE SCREEN command has no effect on dBASE screen formatting files (.scr, .fmt, .fmo).

Example

The following example saves the current screen and then displays a message. After the user presses a key, the original screen image is restored and the program continues.

```
mailfile = USER() + ".box"
IF FILE(mailfile)
   SAVE SCREEN TO mailsave
   @ 10, 10 TO 14, 69 DOUBLE
   @ 11, 11 CLEAR TO 13, 68
   @ 12, 24 SAY "You have new mail. Press any key."
   tempkey = INKEY(0)
   RESTORE SCREEN FROM mailsave
   RELEASE SCREEN mailsave
ENDIF
```

See Also

CLEAR SCREENS, RELEASE SCREENS, RESTORE SCREEN

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 lets you save the windows that you have defined so that you can use them again without redefining them. You can define a set of windows, use them, and then CLEAR them until you need to use them again..

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

ACTIVATE WINDOW, CLEAR, DEFINE WINDOW, RESTORE WINDOW

SCAN

The SCAN command is a looping programming construct with an implicit skip, suitable for applying incremental processing commands to the records of a database file.

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. When a SCAN hits the end of a file (or the FOR or WHILE condition ends), the next command after ENDSCAN is performed.

Since SCAN issues its own record pointer movement commands, there is no need to include a SKIP command.

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.

Use SEEK with SET NEAR ON to find exact matches. Use the FOUND() function to check whether the match was exact or not.

With SET NEAR ON, after a SEEK command FOUND() returns a true (.T.) if an exact match has occurred. FOUND() returns a false (.F.) for a near match, and the record pointer is positioned at the record whose key sorts immediately next to the value being sought.

With SET NEAR OFF, after a SEEK command FOUND() returns a false (.F.) if no exact match occurs. The record pointer is positioned at end-of-file.

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(), SEEK(), 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, unless a relation has been set.

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, SELECT(), 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.

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.). Entering a subsequent SKIP will return an error message.

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.

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(), SET SKIP

SORT

SORT creates a new database file in which the records of the active database file are positioned in the alphabetical, chronological, or numerical 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 uppercase and lowercase. 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 when a character expression determines the 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. The following 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.
- The INDEX command provides a UNIQUE option and a FOR option. UNIQUE allows you to index only on the first occurrence of a number of records with the same key value; FOR allows you to index on a subset of records that satisfy a condition. You can index in either ascending or descending order, but not both.
- The SORT command provides FOR and WHILE options. FOR allows you to sort records that satisfy a condition (for example, FOR Lastname = "Jones"). WHILE allows you to specify a secondary sort condition (for example, WHILE Zipcode = "90311"). A SORT can be in ascending order by the values in one field and in descending order by the values in another field.

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

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 that of a dBASE IV command, or of an abbreviation for a command name.

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 or calculated 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, depending on the source expression.

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, COUNT, DECLARE, STORE

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 choose **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. If you modify the .prg of a SUSPENDed dBASE program, you must cancel and compile the program for your changes to be honored. 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, printer, or file. 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 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 to be 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

INDEX, MODIFY STRUCTURE, SET SAFETY, SORT

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 with IBM extended ASCII 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 IBM extended ASCII 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.

If you include TO FILE, the output will be sent to a disk file and you will be prompted for the filename.

See Also

SET DATE, SET HEADING, SET PATH

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 was 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 obtained. It releases the file lock, if a file lock was the last lock obtained. 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()

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/or .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 <.ndx filename>/[TAG]<.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.

To include information about a database file in a catalog, open the catalog using SET CATALOG TO or SET CATALOG ON and then USE the database file.

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.

You can use an indirect reference for <filename>. An indirect reference is a character expression that evaluates to a filename, and can be used anywhere you are asked to provide a filename. You must use an operator in the expression (usually parentheses) so that dBASE IV knows that the character string is an expression, not the literal filename. An indirect reference is similar to using the macro substitution character, but operates much faster.

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 that, when the index is invoked, present the records in *index order*. It is advisable to reindex all dBASE III PLUS files in dBASE IV.

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 10 .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 to the number specified by the FILES setting in Config.sys or Config.db (whichever number is lower). The number of open files also is limited by available memory. Five file handles are reserved by dBASE IV, leaving a possible maximum of 94.

Options

If the database file has a production .mdx file, both files are opened whenever you USE the database file. To open another .mdx file or an .ndx file, use the INDEX clause followed by a filename. If you do not specify a file extension with the filename in the INDEX clause, dBASE IV searches first for an .mdx file with the same name. Therefore, if you have an .mdx file and .ndx file with the same name, provide the .ndx extension along with the filename to open the .ndx file.

After opening the database file with USE, you can open and close all index files and tags with SET INDEX TO. You can also close .ndx files with CLOSE INDEX and DELETE TAG.

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. The <.ndx filename> parameter of the ORDER clause is used when there is more than one .ndx file associated with the database.

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 10 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.

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 IV reserves for other uses. If a filename contains characters that prohibit it from being used as the default alias, dBASE IV 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).

Examples

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
```

This example opens both an .ndx file and an .mdx file:

```
. USE Friends INDEX Zip ORDER Name
```

This example opens a non-production .mdx file:

```
. USE Phone ORDER Last OF Phonea
```

See Also

&(macro), CLEAR ALL, CLOSE, COPY INDEX, COPY TAG, DBF(), DELETE TAG, INDEX, SELECT, SET CATALOG, SET EXCLUSIVE, SET INDEX, SET ORDER

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 IBM extended ASCII 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.

If, while a WAIT is in progress, SET ESCAPE is ON and you press **Esc**, you will receive an "interrupted" error. Therefore, SET ESCAPE OFF before executing a WAIT.

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.

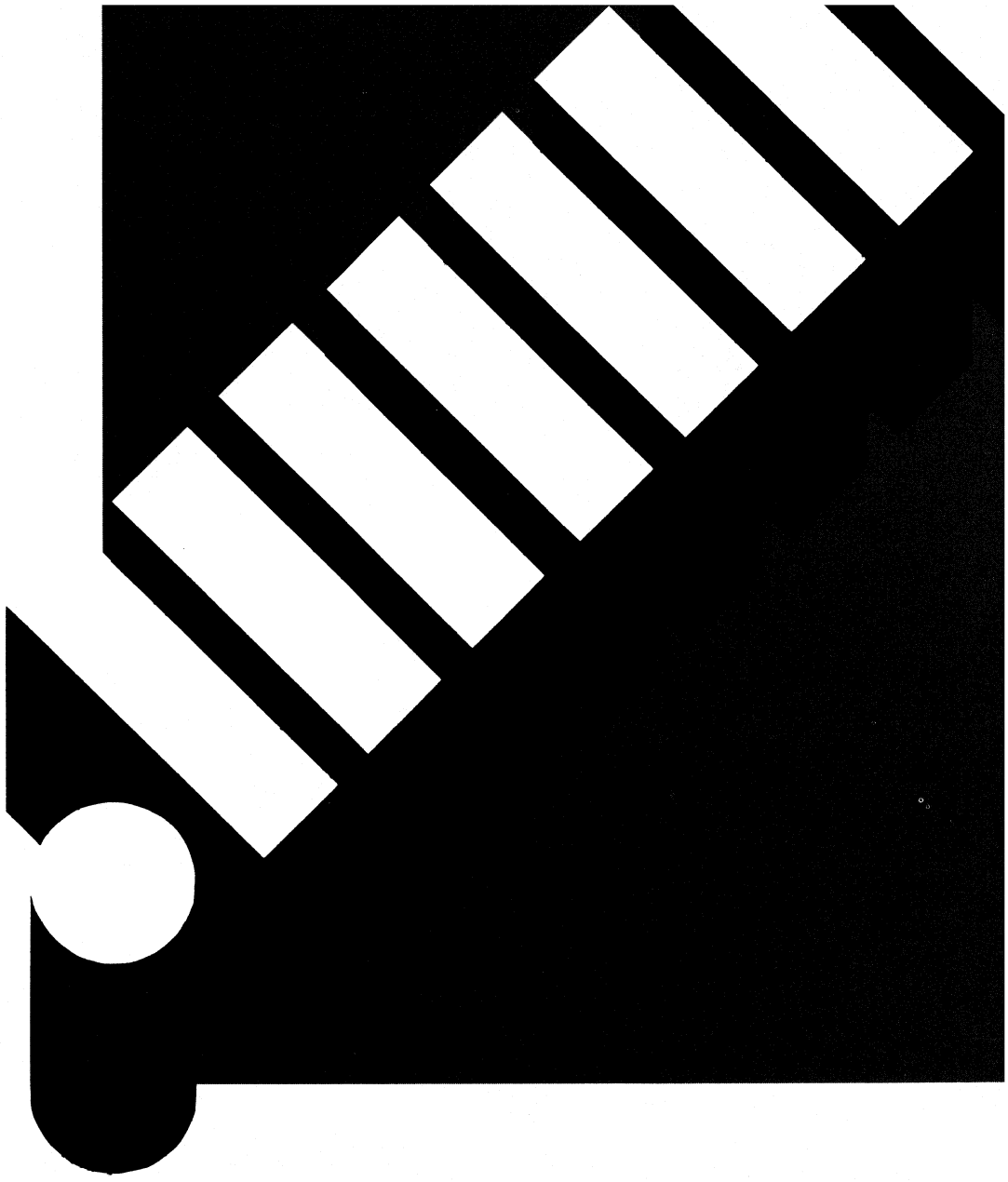
Special Cases

In a network environment, the database file must be in exclusive use before you issue the ZAP command.

See Also

DELETE, PACK, SET SAFETY

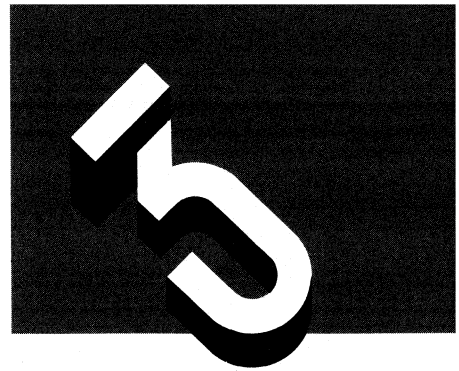
SET Commands



i 1 2 3 4 5 6 7 A B

C D E F G H In

SET Commands



SET

SET displays a menu for 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 see the screen appearance of your selections in a side window as you make them. Press **Ctrl-End** to save 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. Press **Esc** to leave without saving your selections.

Use the **Keys** submenu to program certain function keys and key combinations.

Use the **Disk** submenu to change the default disk drive, drive search path, and operating system working drive and directory.

Use the **Files** submenu to create alternate, format, device, or index file types.

See Also

You can change the default settings of some SET commands in the Config.db file. Refer to *Getting Started* 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 alternate file is written on the default drive, unless a drive designator for another drive is included with the filename. 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 moves the cursor to the end of the alternate file, 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 CLOSE ALTERNATE or SET ALTERNATE TO ↵. 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 first ? 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 IV 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 this text:

Mary had a little lamb

followed by ↵, and save 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."
CLOSE ALTERNATE
```

The file Alt_out.txt now contains:

```
Mary had a little lamb
whose fleece was white as snow.
```

See Also

CLOSE, SET()

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 ON command reduces chances of data loss by saving records to the disk after each change to a record. When SET AUTOSAVE is OFF, the default, dBASE IV saves records to disk as the record buffer is filled.

Commands affected by SET AUTOSAVE are APPEND, APPEND FROM, BROWSE, PACK, READ, REINDEX, REPLACE, and ZAP. With SET AUTOSAVE ON, REPLACE saves each record after the record is changed, even if the scope includes more than one record.

If the database file contains a memo field, SET AUTOSAVE ensures the .dbt file is also updated.

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

SET()

Getting Started

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 1 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 *Getting Started*.

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)
```

See Also

SET()

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 *Getting Started*.

See Also

COPY, COPY INDEXES, CREATE/MODIFY STRUCTURE, INDEX, SET()

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 can be:

- a decimal value for any IBM ASCII character
- a character string
- a memory variable
- a CHR() function

The required order for specifying the sides and the corners using 1 through 8 is illustrated in Figure 3-1.

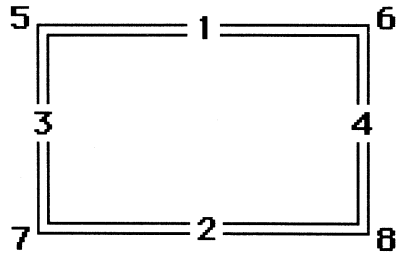


Figure 3-1 Order of border characters

SET BORDER

Default

The default border is a single line box.

Usage

SET BORDER does not affect the borders of objects that have been defined already. You must use SET BORDER before you define a menu, popup, or window for the SET BORDER command to have any effect.

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.

Borders defined explicitly in a DEFINE WINDOW command override the SET BORDER settings.

Examples

To draw a box of asterisks around a portion of the screen:

```
. SET BORDER TO "*"
. @10,15 TO 20,55
```

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
```

or

```
. SET BORDER TO 205, 205, CHR(179), CHR(179), "+", "+", "+", "+"
```

To set a border that is an inverse video bar, enter the command:

```
. SET BORDER TO PANEL
```

See Also

@, @...TO, DEFINE WINDOW

SET CARRY

SET CARRY copies data from the prior record to the new record for all fields, or only specified fields, depending on the syntax you use.

Syntax

SET CARRY on/OFF

SET CARRY TO [<field list> [ADDITIVE]]

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 copied into the new record during APPEND and INSERT.

If you use SET CARRY TO <field list>, only the specified fields are copied during an APPEND, BROWSE or EDIT (when appending new records), or INSERT. A SET FIELDS list overrides a SET CARRY field list. If you provide a field list with a SET FIELDS or BROWSE command, only those fields will be copied.

If you use the ADDITIVE option, the fields in the field list are added to the list of previous fields specified with SET CARRY.

SET CARRY TO ↵ with no field 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 <field list> automatically turns SET CARRY to ON so that the field list can be carried forward.

NOTE
Remember that INSERT works like APPEND if an index is active. For example, if a database file contains ten records, and you want to add a new record number 6 containing the same information as record number 5, you must SET ORDER TO to restore the database to natural order, then SET CARRY ON and INSERT.

See Also

APPEND, BROWSE, EDIT, INSERT, READ, SET(), 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.

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. A master catalog, Catalog.cat, stores catalog filenames along with catalog title descriptions. The description you enter for each catalog entry is displayed later when you use the SET CATALOG TO ? command to select a catalog name.

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.

SET CATALOG

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.

To close a catalog, use SET CATALOG TO ↵. 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 closing the catalog with SET CATALOG TO ↵, keeps the catalog open. 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. With SET CATALOG TO ↵, however, MODIFY REPORT ? returns a list of all .frm files in the current directory.

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.

SET CATALOG

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(), 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(), 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.

See Also

SET()

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*, *enhanced*, *perimeter*, and *background* are areas of the screen that can be assigned different *attributes*, which are described below.

SET COLOR OF
NORMAL/MESSAGES/TITLES/BOX/HIGHLIGHT/INFORMATION/
FIELDS TO <attribute>

An attribute is a character or combination of characters representing color, brightness, blinking, underline, or inverse video display on the screen. On some systems, attributes for both the foreground and background can be set, but the two must be separated with a slash (/). The characters which may be used in an attribute are

W - White	G - Green
N - Black	B - Blue
U - Underline	X - Blank
I - Inverse video	+ - Bright
R - Red	* - Blinking

Attributes are discussed in greater detail under Usage, below.

Usage

There are three forms of the SET COLOR command. Each form is described in one of the following sections.

SET COLOR ON/OFF

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. You can change the setting in the Config.db file with the statement COLOR = ON or COLOR = OFF.

The ISCOLOR() function determines if the current video mode supports color. If ISCOLOR() returns a logical true (.T.), the system is usually equipped with a color card and should be able to display color.

SET COLOR

SET COLOR TO [<standard>],[<enhanced>],[<perimeter>],[<background>]

Each time you start dBASE IV, any color settings in the Config.db file determine the screen appearance. Refer to *Getting Started* for information on setting colors in the Config.db file.

If you type SET COLOR TO ↵, without providing any color settings, dBASE IV resets the screen to black and white. This command does not reset the colors to the Config.db definitions.

In place of <standard>, <enhanced>, <perimeter>, and <background> in the syntax above, you provide attributes indicating the colors that should appear when that area is displayed.

Standard and *enhanced* refer to areas of the screen that can be set on every type of monitor. These areas are defined in greater detail in Table 3-4. The standard areas are NORMAL, MESSAGES, and TITLES, and include most of the basic display areas of dBASE IV. Enhanced areas, which include BOX, HIGHLIGHT, INFORMATION, and FIELDS, are given special emphasis on the screen. For example,

```
. SET COLOR TO B, G
```

sets the text that appears in the standard areas to blue, and the text that appears in the enhanced areas to green. For better visibility on the screen, however, you may want to set the color against which the text appears. The color of the text is called the foreground, the color against which the text appears is called the background. You can usually supply both a foreground and background color for the standard and enhanced areas. Separate the foreground and background colors with a slash (/). For example, the following command gives you white text (white foreground) against a blue background for standard areas, and black text (black foreground) on a green background for the enhanced areas:

```
. SET COLOR TO W/B, N/G
```

Perimeter and *background* only apply to certain monitors. The perimeter borders the area that displays text on the screen. On some display systems, such as the enhanced graphics adapter (EGA or compatible), the perimeter color cannot be set. The next command sets the standard areas to white on blue, the enhanced areas to black on green, and the perimeter to red:

```
. SET COLOR TO W/B, N/G, R
```

Setting the background color with this third parameter only works with display systems that require the same background for the standard and enhanced areas. On these systems, set the foreground colors for standard and enhanced, then the background color that will provide the background for both the standard and enhanced areas.

```
. SET COLOR TO W, N, R, B
```

sets the standard areas to white on blue, the enhanced areas to black on blue, and the perimeter to red. On most systems, this command would be entered

```
. SET COLOR TO W/B, N/B, R
```

Notice that the syntax for SET COLOR TO requires that the standard, enhanced, perimeter, and background areas are always provided in this exact order. Therefore, to change the enhanced area to black foreground on a green background, but leave the standard area at its original color setting, type:

```
. SET COLOR TO ,N/G
```

The comma indicates that the first area is omitted, so the standard setting is unchanged. To change just the perimeter to blue:

```
. SET COLOR TO ,, B
```

The characters used to form an attribute may be uppercase or lowercase. Not all attributes will work on all display systems, so you should test which combinations work on your system. For example, although you can enter an attribute to set underlining or inverse video on a color system, these are intended for use on a monochrome system, and underlines and inverse video will not appear on your color display. Perimeters, as stated above, cannot be set on all systems.

W (white), N (black), I (inverse video), and U (underlining) work on a monochrome monitor.

In addition to W and N, color systems can be set to RGB colors. Combinations of R (red), G (green), and B (blue) produce different colors, and the symbols can appear in any order. RB, BR, rb, and br all indicate the color magenta. Table 3-2 summarizes the color attributes you can use with SET COLOR.

Table 3-2 Color symbols

Foreground and background		Foreground only	
Symbol	*Color*	*Symbol*	*Color*
W	White	W+	Bright white
N	Black	N+	Grey
R	Red	R+	Pink
G	Green	G+	Light green
B	Blue	B+	Light blue
RG (or GR)	Brown	RG+ (or GR+)	Yellow
RB (or BR)	Magenta	RB+ (or BR+)	Bright magenta
GB (or BG)	Cyan	GB+ (or GB+)	Light cyan
RGB	White	RGB+	Bright white

Note that the + attribute, which indicates brightness of color, affects only the foreground colors. Even if you place the plus sign (+) in the background attribute, the foreground, not the background, displays bright. The following command sets the foreground to bright white, and the background to blue:

```
. SET COLOR TO W/B+
```

X is the same as a black foreground on a black background, or N/N, and may be used if you do not want characters in a specific area to appear on your screen. For example,

```
. SET COLOR TO ,X
```

hides the password that you enter in a GET. Using an asterisk (*) in an attribute indicates blinking display. As with the plus sign (+), the asterisk applies only to the foreground. Therefore, only the foreground will blink, even if you place the asterisk along with a background color setting.

```
. SET COLOR TO W/B+*
```

sets a blinking bright white foreground on a blue background.

Table 3-3 summarizes special attribute characters of the SET COLOR command.

Table 3-3 Attribute characters

Symbol	Note
I	Inverse Video, in monochrome only
U	Underlining, in monochrome only
X	Black foreground on black background, suppresses an area's display
+	Bright, applies to foreground color only
*	Blinking, applies to foreground color only

SET COLOR OF NORMAL/MESSAGES/TITLES/BOX/HIGHLIGHT/INFORMATION/ FIELDS TO <attribute>

As shown in Table 3-4, NORMAL, MESSAGES, TITLES, BOX, HIGHLIGHT, INFORMATION, and FIELDS are names assigned to a group of predefined screen displays. The first three, NORMAL, MESSAGES, and TITLES, are collectively known as the standard area. The last four, BOX, HIGHLIGHT, INFORMATION, and FIELDS, form the enhanced area. Therefore, when you change the colors of the standard area, you are also changing all the colors of NORMAL, MESSAGES, and TITLES. With this third syntax of the SET COLOR command, you can control each group within the standard and enhanced areas. You can, for example, change the MESSAGES group, as listed in Table 3-4, without changing NORMAL or TITLES.

Table 3-4 Screen areas

Standard Area Groups	Displays Affected
NORMAL	@...SAY output, without the COLOR keyword Unselected fields in BROWSE Layout editor design surface (in report, label, form design) [1] Unselected text on design surface Unselected uncolored display only field templates (in report, label, form design) Static memo, window borders (in report, label, form design) Unselected conditions in a QBE file skeleton Unselected conditions in a QBE condition box Calculated field expressions in QBE Unselected, uncolored box borders in layout editor (in report, label, form design)

(continued)

Table 3-4 Screen areas (*continued*)

Standard Area Groups	Displays Affected
NORMAL (*continued*)	Uncolored box borders drawn with @...TO command User-entered text in the Applications Generator Scoreboard area Database design grid (CREATE and MODIFY STRUCTURE) Database design column labels (CREATE and MODIFY STRUCTURE) Database design field numbers (CREATE and MODIFY STRUCTURE)
MESSAGES	Message line messages [2] Error messages in an error box [2] Cited command line in an error box [3] Navigation line messages [2] Available, unselected menu and list choices [2] Unavailable menu and list choices [3] File window contents[2] Help box interiors Bolded Help box text [2] Prompt box interiors Unselected prompt box buttons [3] Unselected error box buttons [3] Unselected Help box buttons [3]
TITLES	LIST/DISPLAY headings Help box headings Control Center banner [2] Control Center catalog name heading and filename Control Center filename and file description headings BROWSE field name headings BROWSE table grids [3] Directory name in a file pick list Unselected report design band lines QBE file skeleton field names and filename QBE view skeleton field names and view name QBE view skeleton grid [3] QBE file skeleton grids [3] QBE calculated fields skeleton grids [3] QBE condition box border [3] QBE condition box heading Ruler line (in word wrap editor: MODIFY COMMAND, memo field editor)

(*continued*)

Table 3-4 Screen areas (*continued*)

Standard Area Groups	Displays Affected
TITLES *(continued)*	Text styles defined with the **Style** menu in layout editor[2] End-of-Report line in report design Static text in Applications Generator forms Underlined Help box text [2]

Enhanced Area Groups	Displays Affected
BOX	Menu borders File and sorted by information in **DOS Utilities** File window border in **DOS Utilities** and other lists Pick list borders Prompt box borders Label design layout box border Error box borders Unselected Applications Generator object border
HIGHLIGHT	Highlighted menu and list choices [2] Selected static text on design surface (in report, form, label design) Selected field on design surface (in report, form, label design) Selected box (in report, form, label design) Control Center filename and file description Flying cursor in ruler line (design surfaces and word wrap editor) Report design band line labels Report design selected band lines Information box borders Information box interiors Current field in the Applications Generator Selected button in Help box [3]
INFORMATION	Clock Help box borders Status line Selected button in error box Highlighted prompt box buttons[3]

(*continued*)

Table 3-4 Screen areas (*continued*)

Standard Area Groups	Displays Affected
FIELDS	Prompt box data entry areas [3] Selected field in BROWSE [3] Editable fields in @...GET Selected fields in database design surface Unselected fields in the Applications Generator

[1]The + attribute, which indicates a bright color, is ignored in this area. Bright colors appear as their corresponding dimmer colors, so that yellow (GR+) appears as brown (GR). See Table 3-2 for colors and brightness.

[2]Colors in this area are always bright, whether you use the bright attribute (+) or not. For example, black (N) appears as grey (N+). See Table 3-2 for colors and brightness.

[3]This area may be bright or dim. For contrast, however, it is better not to use a bright attribute for this area, so that unavailable options will appear dim. If you set this area bright, all options (both available and unavailable) will appear bright.

Table 3-4 shows that the brightness of some areas cannot be set.

Special Cases

The older Compaq portable computers with built-in monochrome monitors, many laptop models, and some other systems function as though they have color monitors. On these computers, monochrome attributes U and I do not generate underlining and inverse video. The color settings are translated to shades of black and green or black and amber.

When you use the DEFINE WINDOW command without the COLOR keyword, the window takes its colors from the current color settings. Without the COLOR keyword, the window's frame will be the current color of the BOX subgroup. You may also use the COLOR keyword of DEFINE WINDOW to designate new colors. Whether you set the window's colors explicitly with the COLOR keyword of DEFINE WINDOW, or the window colors default to the current SET COLOR settings, once the window is DEFINEd, those colors are part of the definition. If you later change the screen colors with SET COLOR and then ACTIVATE WINDOW, the window has the colors it was defined with, not the new color settings.

Examples

To set the standard areas to bright white letters on a blue background, and an enhanced display to yellow letters on a black background, at the dot prompt type:

```
. SET COLOR TO W+/B, GR+/N
```

The status line is part of the INFORMATION subgroup. To set the status line to bright white on red, type:

```
. SET COLOR OF INFORMATION TO W+/R
```

Standard areas remain bright white on blue, and enhanced areas remain yellow on black. The clock, like the status line, is bright white on red, as this area is also part of the INFORMATION group.

See Also

@, @...FILL, DEFINE WINDOW, ISCOLOR(), SET, 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.

See Also

SET()

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.

See Also

SET()

SET CURRENCY

The SET CURRENCY command changes the symbol used for currency.

Syntax

SET CURRENCY TO [<expC>]

Usage

When used with PICTURE or FUNCTION clauses, the SET CURRENCY command changes the currency symbol that is displayed. dBASE IV allows a maximum of nine characters to be used for this symbol.

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
```

See Also

@, SET CURRENCY LEFT/RIGHT, SET POINT, SET SEPARATOR

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 *Getting Started.*.

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
```

See Also

@, SET CURRENCY, SET POINT, SET SEPARATOR

SET CURSOR

SET CURSOR determines if the cursor is displayed.

Syntax

SET CURSOR ON/off

Default

The default for SET CURSOR is ON.

Usage

When SET CURSOR is OFF, the cursor is hidden. Use SET CURSOR ON to redisplay the cursor.

When you use the RUN command, dBASE IV returns the cursor to its state prior to invoking dBASE IV. When the external program returns to dBASE IV, dBASE IV again hides the cursor if SET CURSOR is OFF.

See Also

RUN, RUN(), SET(), SET CONSOLE

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 *Getting Started* for customizing the Config.db file.

The date formats available to the SET DATE command are:

AMERICAN (or MDY)	mm/dd/yy
ANSI	yy.mm.dd
BRITISH or FRENCH (or DMY)	dd/mm/yy
GERMAN	dd.mm.yy
ITALIAN	dd-mm-yy
JAPAN (or YMD)	yy/mm/dd
USA	mm-dd-yy

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 DBTRAP

SET DBTRAP provides a net of protection against program errors when you interrupt the execution of one command to execute a second command.

Syntax

SET DBTRAP ON/off

Usage

A dBASE IV command is interrupted when its execution is temporarily halted in order to execute another dBASE IV command.

If you interrupt the execution of a command, change the environment in which the command is executing, and then continue the command's execution, dBASE IV may not be able to resolve the conflict. For example, if you BROWSE a database file containing a calculated field that calls a UDF, and the UDF tries to USE another database file in the same work area, the original environment would be impossible to return to after the interruption, since the first database file would have been closed. Therefore, UDFs and some of the ON commands require special attention because they generally interrupt command execution in order to execute other, intervening, commands.

If SET DBTRAP is ON, dBASE IV provides four levels of protection against changes to the environment these intervening commands might cause.

1. SET DBTRAP ON bars intervening commands that would remove or change the structure of either the active database file or the active window. Error messages appear if a UDF or ON command includes commands that attempt a barred activity. Specifically:

   a. Intervening commands cannot CLOSE, PACK, MODIFY STRUCTURE, USE, or ZAP the database file that was active when the interruption occurred.

   b. Intervening commands cannot CLEAR WINDOW, DEFINE WINDOW, RELEASE WINDOW, or RESTORE WINDOW with a window name that was active when the interruption occurred.

2. SET DBTRAP ON saves some aspects of the current environment before executing intervening commands, and restores this environment when the intervening commands are complete. If you change these aspects of the environment, no error message appears, but your changes will only be in effect while the intervening commands are executing. Once their execution is complete, the changes are reset.

   Specifically, SET DBTRAP ON saves the work area and window (if any) that was active at the time of the interruption. Even if intervening commands SELECT a new work area, DEACTIVATE the active WINDOW, or DEFINE a new WINDOW, SET DBTRAP ON restores the original work area and window when you return from the UDF or ON command interruption.

3. SET DBTRAP ON prohibits a recursive BROWSE or EDIT. For example, if you BROWSE a calculated field that calls a UDF, and the UDF contains another BROWSE (or an EDIT, CHANGE, INSERT, or APPEND), the BROWSE was recursive. APPEND, BROWSE, CHANGE, EDIT, and INSERT are closely related commands, so any one of these commands is prevented from calling itself or any other command.

   With SET DBTRAP ON, the restriction on recursive BROWSE applies to any work area. An EDIT in one work area cannot call an APPEND in another work area. With SET DBTRAP OFF, however, the restriction on recursive BROWSE applies only to the current work area. An EDIT can call an APPEND, if the file to be APPENDed is open in another work area.

   The LIST and DISPLAY commands cannot be recursively called whether DBTRAP is ON or OFF. dBASE IV prevents you from LISTing a database file and interrupting the LIST to execute another LIST or DISPLAY. LIST and DISPLAY are also closely related, and are also mutually barred from recursive calls.

4. With SET DBTRAP ON, a UDF cannot be used in either the index expression or FOR clause of an INDEX command. Interrupting the construction of an index may create situations that INDEX cannot easily resolve.

With SET DBTRAP OFF, these restrictions are lifted (except the use of recursive LIST/DISPLAY, and of BROWSE-type commands in the same work area, as noted above).

WARNING
Do not SET DBTRAP OFF unless you are aware of the possible results, and are either sure these results will not affect your program or are providing other safeguards if an error occurs.

UDFs and ON commands increase the possibility of programming errors, whether SET DBTRAP is OFF or ON, and you must pay particular attention to the logical control flow.

See Also

FUNCTION, ON ERROR/ESCAPE/KEY

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(), SET ECHO, SET PRINTER, 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 digits 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
```

See Also

SET()

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 or is ready and does not change the drive to which you return when you quit dBASE IV.

Refer to *Gettting Started* to find out how to set the default drive in the Config.db file.

SET DIRECTORY overrides SET DEFAULT and changes the default drive to the new working drive. SET DIRECTORY is preferred over SET DEFAULT because it changes both the drive and directory in one step, and it ensures that your dBASE IV drive and directory match your operating system drive and directory.

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(), SET DIRECTORY, 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

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. If you position the record pointer on a specific record, as with the GO command, that record appears whether or not it is marked for deletion.

INDEX and REINDEX include records marked for deletion, even if SET DELETED is ON.

If SET DELETED is ON, RECALL ALL does not recall any records, because the deleted records are ignored.

If two files are related with SET RELATION, SET DELETED ON suppresses the display of deleted records in the child file. The related record in the parent file, however, still appears (unless, of course, the parent record is also deleted).

Example

You must move the record pointer away from the deleted record for SET DELETED to take effect. For example,

```
. GO 2
. DELETE
. SET DELETED ON
. EDIT
```

will allow you to edit the deleted record number 2, because you did not move the record pointer after SET DELETED ON.

See Also

DELETED(), SET(), SET FILTER, SET RELATION

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(), SET INTENSITY

SET DESIGN

SET DESIGN prevents you from creating or modifying certain files from the Control Center, from the dot prompt, or in a procedure.

Syntax

SET DESIGN ON/off

Usage

This command allows an applications developer to prevent the user from entering design mode (that is, creating or modifying files from the **Data**, **Queries**, **Forms**, **Reports**, **Labels**, and **Applications** panels) in the Control Center. The developer can thereby use the Control Center menu system as the interface for a limited application. SET DESIGN OFF also disables the **Shift-F2** key combination which would otherwise take you to the QBE designer from APPEND, BROWSE, or EDIT.

From the dot prompt or in a procedure, SET DESIGN OFF prevents you from using CREATE or MODIFY STRUCTURE, CREATE/MODIFY APPLICATION, CREATE/MODIFY LABEL, CREATE/MODIFY QUERY/VIEW, CREATE/MODIFY REPORT, CREATE/MODIFY SCREEN, and MODIFY COMMAND/FILE. You also cannot enter the QBE designer from BROWSE or EDIT.

You can, however, still CREATE VIEW FROM ENVIRONMENT.

If SET DESIGN is ON, which is the default setting, you can create or modify any of these files.

See Also

CREATE or MODIFY STRUCTURE, CREATE VIEW FROM ENVIRONMENT, CREATE/MODIFY APPLICATION, CREATE/MODIFY LABEL, CREATE/MODIFY QUERY/VIEW, CREATE/MODIFY REPORT, CREATE/MODIFY SCREEN, MODIFY COMMAND/FILE, SET()

SET DEVELOPMENT

SET DEVELOPMENT triggers an automatic compilation of program source (.prg) files if you use an external text editor.

Syntax

SET DEVELOPMENT ON/off

Default

The default for SET DEVELOPMENT is ON.

Usage

SET DEVELOPMENT has no effect unless you set up an external text editor with TEDIT in Config.db.

If you do not set TEDIT in the Config.db file, MODIFY COMMAND will call the dBASE IV internal text editor. If you edit an existing program source (.prg) file with the internal editor, the associated object (.dbo) file is erased. When you later DO the program file, DO notes that there is no corresponding .dbo file, and compiles a new object file before executing the program.

Using an external editor with TEDIT, however, does not automatically delete the outdated .dbo file. If SET DEVELOPMENT is OFF, the file is not recompiled by a subsequent DO command, and the source and object files do not match.

With an external editor, SET DEVELOPMENT ON forces compilation of a the .dbo file after a MODIFY COMMAND. With SET DEVELOPMENT ON, DO compares the creation dates of the compiled object (.dbo) file with its program source (.prg) file. If the object file is older than the source file, DO compiles a new object file.

SET DEVELOPMENT OFF disables the trigger, so DO will not check for an outdated .dbo file. If the program file was edited with an external editor and SET DEVELOPMENT is OFF, DO may execute the outdated .dbo file.

See Also

COMPILE, DO, MODIFY COMMAND/FILE, SET()

SET DEVICE

SET DEVICE determines whether @...SAY output is 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 DIRECTORY

SET DIRECTORY specifies the operating system working drive and directory from within dBASE IV.

Syntax

SET DIRECTORY TO [[<drive:>][<path>]]

Defaults

SET DIRECTORY TO with no path restores the working drive and directory to the startup drive and directory.

When SET DIRECTORY sets the operating system working drive, the dBASE IV default drive is also set. However, SET DEFAULT does not change the operating system working drive.

When you quit dBASE IV, the working drive and directory are restored to the startup drive and directory.

Usage

<drive:> and <path>form an operating system path name or a character expression that evaluates to a path name. This can be a drive identifier (include the colon), a directory name, or both. Do not leave spaces between the drive identifier and directory name. Do not end the directory name with a slash.

This command provides the functionality of the RUN CD <path>, RUN <drive:>, and SET DEFAULT TO <drive:> commands. SET DIRECTORY is preferred because you can change the dBASE IV default drive and directory and the operating system drive and directory in one step.

If only a drive is specified, the working directory becomes the last directory associated with that drive (or the root directory, if none has been associated).

During compilation, if [[<drive:>][<path>]] is a constant string, all characters until the first space are considered to be the path. The path is not validated as a legal operating system path during compilation.

Refer to *Getting Started* to find out how to set the working drive and directory in the Config.db file.

SET DIRECTORY

Examples

The following command changes the working drive and directory to E:\ACCT:

```
. SET DIRECTORY TO E:\ACCT
E:\ACCT
```

The following command changes the working directory to C:\SAMPLES:

```
. m_path = "C:\SAMPLES"
\SAMPLES
. SET DIRECTORY TO &m_path
```

See Also

DIR, RUN, RUN(), SET(), SET DEFAULT, SET PATH

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 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.

Example

If an external program might reset the display mode, you can save the current display mode to a memory variable and then SET DISPLAY upon returning to dBASE IV.

```
. xmode = SET("DISPLAY")
. RUN <external program>
. SET DISPLAY TO &xmode
```

See Also

SET, SET(), SET COLOR

SET ECHO

SET ECHO displays command lines from dBASE IV programs on the screen 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 SET 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(), 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.

Syntax

SET ENCRYPTION ON/off

Default

The default for SET ENCRYPTION is ON

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 use EXPORT, COPY STRUCTURE EXTENDED, MODIFY STRUCTURE, or 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.

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.

You can encrypt any newly created database file by assigning the file an access level through PROTECT.

See Also

PROTECT, SET()

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 **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. With SET ESCAPE OFF, ON ESCAPE and ON KEY (without a key label) will not trap program interuptions. SET ESCAPE OFF also disables the operation of **Ctrl-S** 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 ERROR/ESCAPE/KEY, READKEY(), SET()

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 are the same up to that point, the result of the comparison is true (.T.).

If SET EXACT is ON, the two strings must be identical, except for trailing blanks, for the comparison to be evaluated as true (.T.).

dBASE IV evaluates constants during compilation, and does not re-evaluate them during program execution. If this command line

? "abc" ="a"

is included in a program and compiled with SET EXACT OFF, it returns a true (.T.) because only the first character in each string was compared. Even if you later SET EXACT ON,

? "abc" = "a"

continues to return a .T. in the program.

SET EXACT

Examples

Comparing two constant string expressions produces different results, depending on whether SET EXACT is ON or OFF. For example:

```
. SET EXACT ON
. ? "abc" = "a"
.F.
. ? "a" = "abc"
.F.
. ? "abc" = "abc"
.T.
. SET EXACT OFF
. ? "abc" = "a"
.T.
. ? "a" = "abc"
.F.
. ? "abc" = "abc"
.T.
```

See Also

SET()

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.

The CREATE command executes a temporary SET EXCLUSIVE ON until the database file is written.

See Also

COPY INDEXES/TAG, INDEX ON, PROTECT, SET(), SET ENCRYPTION, USE

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 field 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 identifier>...]
[,<field>[/R] /<calculated field identifier>...]

SET FIELDS TO ALL [LIKE/EXCEPT <skeleton>]

Default

The default for SET FIELDS is OFF. It is set ON when you specify a field list with SET FIELDS TO. Do not use SET FIELDS ON without first specifying a field 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 fields only.

The calculated field identifier 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, all fields in 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 ↵ clears the field list. SET FIELDS TO ALL includes all the fields of the active database file.

After a field list is set, additional or consecutive SET FIELDS TO commands add fields to the current field list. To include fields from database files in other work areas, SELECT the work area of the file and use the SET FIELDS 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 field 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. The SET FIELDS TO command does not check to see if files are related; you must verify this yourself.

Affected Commands

The field list that you define with the SET FIELDS command is used by all commands that have a field list option in their syntax. With SET FIELDS ON, the following commands use the field list specified by SET FIELDS TO:

APPEND
AVERAGE
BROWSE
CALCULATE
CHANGE
CREATE/MODIFY VIEW FROM ENVIRONMENT
COPY TO
COPY STRUCTURE
COPY TO ARRAY
DISPLAY
EDIT
EXPORT
JOIN
LIST
SUM
TOTAL

SET CARRY also uses the SET FIELDS list.

If you establish a field list and then COPY STRUCTURE, the new structure contains only those fields in the field list. Clear the field list with SET FIELDS TO ↵ before APPENDing records to the new database file.

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.

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 field 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 field 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 NEXT 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.

```
. Use Client
. SET FIELDS TO ALL LIKE *name
. LIST

Record#      LASTNAME       FIRSTNAME
      1      Wright         Fred
      2      Bailey         Sandra
      3      Martinez       Ric
      4      Peters         Kimberly
      5      Yamada         George J.
      6      Timmons        Gene
      7      Maxwell        Florence
      8      Beckman        Riener
```

See Also

CLEAR FIELDS, CREATE/MODIFY VIEW, CREATE VIEW FROM ENVIRONMENT, SET(), SET CARRY, 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>]

Defaults

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 dBASE III PLUS query file created by CREATE/MODIFY QUERY. You cannot use a dBASE IV query file (.qbe) with this command. 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 = {01/01/91}. Both conditions can also be present together, SET FILTER TO lastname = "Jones" .AND. Departure >{01/01/91}.

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(), SET DELETED

SET FORMAT

SET FORMAT allows a form contained in a format (.fmt) file to be used with the READ, EDIT, APPEND, INSERT, CHANGE, and BROWSE commands.

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 when you are editing or changing records in the associated database file.

The SET FORMAT command activates the format file named in the command for use with full screen editing commands.

The SET FORMAT 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. 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.

dBASE III PLUS format files that have an .fmt extension are compiled to .fmo by the dBASE IV SET FORMAT command.

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.

NOTE
If you open a format file in a program, then issue an @...GET followed by a READ, the READ will get the format file. Close the format file after using it in a program to permit other variables to be read.

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 is not used, APPEND, CHANGE, EDIT, and INSERT use the standard display and entry form dBASE IV provides.

The FORMAT option of the BROWSE command applies all @...GET options contained in the format file (except the position of fields on the screen) to the BROWSE table.

The SET FORMAT 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, BROWSE, CHANGE, CLOSE, EDIT, INSERT, READ, SET(), SET DEVICE

SET FULLPATH

SET FULLPATH permits or suppresses the display of a full pathname with the NDX(), MDX(), and DBF() functions.

Syntax

SET FULLPATH on/OFF

Default

The default for SET FULLPATH is OFF.

Usage

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.

Use this command in programs that use the NDX(), MDX(), or DBF() functions to make their output compatible with dBASE III PLUS.

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.

See Also

SET()

See *dBASE IV Change Summary* for a discussion of changes and enhancements introduced in dBASE IV.

SET FUNCTION

SET FUNCTION programs a 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 *Getting Started*.

The standard function key assignments are listed in Table 3-5.

Table 3-5 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.

You may assign up to 254 characters to each function key.

Function key assignments are overwritten in BROWSE and EDIT. Use an ON KEY command followed by DO, PLAY MACRO, or KEYBOARD to output a text string in BROWSE/EDIT.

NOTE
If SET FUNCTION and ON KEY assign different commands to the same key, the ON KEY assignment overrides that of SET FUNCTION.

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;"
```

SET HEADINGS

SET HEADINGS determines whether titles are shown for AVERAGE, DISPLAY, LIST, SUM, and TYPE.

Syntax

SET HEADINGS ON/off

Default

The default for SET HEADINGS is ON.

Usage

The commands AVERAGE, CALCULATE, DISPLAY, LIST, and SUM display a column title for each displayed field, memory variable, or expression. The TYPE command displays the filename, date, and page number on each page of output.

The column width for fields is the length of the heading or field width, whichever is larger.

Example

```
. USE Transact
. DISPLAY

Record#   Client_id   Order_id   Date_trans   Invoiced   Total_bill
      1   A10025      87-105     02/03/87     .T.          1850.00
          .
          .
          .

. SET HEADINGS OFF
. DISPLAY

          1   A10025      87-105     02/03/87     .T.          1850.00
              .
              .
              .
```

See Also

SET()

SET HELP

SET HELP determines whether a prompt offering dBASE IV Help appears in the pop-up window 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 SET HELP is ON and you make an error entering a command, **CANCEL**, **EDIT**, and **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()

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()

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 *Getting Started*.

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(), 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 <.ndx filename> / [TAG] <.mdx tag> [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.

NOTE
If you have an .ndx file and an .mdx file with the same name, you must specify the .ndx file extension in the SET INDEX statement for dBASE IV to open the .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 and .mdx files in the current work area except the production .mdx file. It is an alternate syntax for CLOSE INDEX.

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.

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 ↵, 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 10 .mdx and .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
```

Create a tag in a new, non-production .mdx file, then make this tag the controlling index. Type:

```
. USE Client
. INDEX ON Lastname + Firstname TAG Fullname OF Clnt2
. SET INDEX TO Cus_name, Clnt2 ORDER Fullname
```

See Also

CLOSE, COPY INDEXES, DELETE TAG, INDEX, KEY(), MDX(), NDX(), ORDER(), REINDEX, SET ORDER, TAG(), USE

SET INSTRUCT

SET INSTRUCT enables the display of prompt boxes or dBASE code as forms, reports, or labels are generated.

Syntax

SET INSTRUCT ON/off

Default

The default for SET INSTRUCT is ON.

Usage

If SET INSTRUCT is ON, this command displays a prompt box of actions at the Control Center when a file is selected.

If SET INSTRUCT is OFF and SET TALK is ON, dBASE code displays on the screen as it is generated during the CREATE/MODIFY LABEL, CREATE/MODIFY REPORT, or CREATE/MODIFY SCREEN operations.

See Also

SET()

SET INTENSITY

SET INTENSITY determines whether inverse video, on monochrome monitors, or COLOR OF FIELDS, on color monitors, is used for full-screen editing commands.

Syntax

SET INTENSITY ON/off

Usage

On a color system, SET INTENSITY ON displays the data entry areas in a full-screen editing operation, such as @...GET, EDIT, or APPEND, with the SET COLOR OF FIELDS attribute. On a monochrome system, these full-screen commands display the data entry areas in inverse video.

If SET INTENSITY is OFF, the data entry areas appear with the same attribute as SET COLOR OF NORMAL, which is the attribute used by @...SAY. On a monochrome system, the data entry areas appear in standard video, not inverse video.

When SET INTENSITY is OFF, you may want to use SET DELIMITERS to set off the data entry areas, because these areas are indistinguishable from the rest of the display.

See Also

SET, SET(), SET COLOR, SET DELIMITERS, SET DISPLAY

SET LOCK

SET LOCK determines whether certain dBASE IV commands automatically lock database files in a multi-user system.

Syntax

SET LOCK ON/off

Usage

Database files are automatically locked to preserve the integrity of your data. When you have a file lock, other users have read-only access; that is, they can read the file, but cannot write to it. Files are automatically locked by certain commands that read and summarize records of the database file, if SET LOCK is ON.

With SET LOCK ON, these commands automatically attempt to lock the file, even though their operation is read-only, so that the summarization of all records in the database file is unaffected by additions and deletions from other users. SET LOCK OFF disables the automatic locking by subsequent summarization commands that you issue.

For example, you might use the REPORT FORM command to get a rough report from a database file. REPORT FORM automatically locks the database file to ensure that updates and deletions from other users do not affect your result. For your purposes, however, you can SET LOCK OFF before using REPORT FORM. With SET LOCK OFF, your operation does not prohibit other users from writing to the database file. Because you just need a rough report, updates from other users will not seriously affect your result. You can then SET LOCK ON for your final report.

Commands affected by SET LOCK like REPORT FORM are AVERAGE, CALCULATE, COUNT, LABEL FORM, and SUM.

Some commands that use automatic locking, such as TOTAL, read information from one database file and create a second file. For these commands, SET LOCK OFF disables the lock on the source file (which is being read), but not on the target file (which is being written). When you create a new file, the file is always locked for your exclusive use.

Commands affected by SET LOCK like TOTAL are COPY INDEXES, COPY STRUCTURE, COPY TAG, JOIN, and SORT. With these commands, SET LOCK OFF prevents the automatic lock of the source file, but not of the target file. When execution of these commands is complete, the target file is unlocked.

See Also

FLOCK(), SET()

Networking with dBASE IV discusses file locks, record locks, and automatic locking.

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

The numeric expression is the number of character positions from the left of the page. SET MARGIN sets the margin for all printed output from dBASE IV.

The SET MARGIN value is the same as the system memory variable __ploffset, unless __ploffset is declared a private memory variable to a procedure. If you declare __ploffset PRIVATE, the original SET MARGIN or __ploffset value is restored when you return from the procedure.

SET MARGIN and __ploffset are not additive.

```
SET MARGIN TO 10
__ploffset = 10
```

sets the left margin to 10 characters, not 20.

You can include the MARGIN = setting in the Config.db file, to assign an initial value to the printer left offset. See *Getting Started*.

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 SET MARGIN are not additive.

SET MARGIN

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.

See Also

SET()

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 255. You may customize this value in the Config.db file. See *Getting Started*.

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

?/?? (@V FUNCTION), MEMLINES(), MLINE(), SET()

SET MESSAGE

SET MESSAGE displays a user-defined message on the screen.

Syntax

SET MESSAGE TO [<expC> [AT <expN>[,<expN>]]]

Usage

The message can have a maximum of 79 characters.

Although a message from @...GET commands, POPUPs, and MENUs overrides the message text defined with SET MESSAGE TO, the output location is still determined by the AT clause. If you use the AT clause and do not specify a column position, the message is centered on the specified line.

If SET STATUS is OFF, SET MESSAGE TO <expC> AT <expN>, <expN> displays a message on the specified row and, optionally, column number only during full-screen editing commands (such as APPEND, EDIT and READ) and when menus are active. The first <expN> in the AT clause specifies the row and is required if the AT clause is used. The second <expN> specifies a column position at which the message text begins. The row position can be between 0 and the last line of the screen. The column position can be between 0 and 79.

If SET STATUS is ON, the AT clause is disregarded, and your message text is displayed centered at the bottom line of the screen. This line is called the *message line*. Messages on the message line are always displayed, not just during full-screen commands.

SET MESSAGE TO ↵ clears a user-defined message.

Tips

Messages display across the full screen, not inside windows. Avoid overwriting a window or other text on the screen with a message line.

SET SCOREBOARD OFF before displaying a message on line 0.

See Also

SET SCOREBOARD, SET STATUS

SET NEAR

SET NEAR moves the record pointer to the record immediately following the sought-after key when FIND, SEEK, or SEEK() do not find an exact match.

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 by FIND, SEEK, or SEEK(), but 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 restrictions.

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.

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
      6     C00002
      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.

See Also

EOF(), FIND, FOUND(), SEEK, SEEK(), SET()

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.

See Also

SET()

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. If the database file has a production .mdx file, you must use the SET ORDER TO TAG form of the command. 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.

Syntax

SET PATH TO [<path list>]

Usage

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. SET PATH TO ↵ removes an active path list.

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.

The default directory is where you started dBASE IV from, unless you changed directories using an operating system command, used SET DEFAULT to specify a different drive, or used SET DIRECTORY to specify a different drive and directory.

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, or use SET DIRECTORY.

Example

dBASE IV first searches the current drive and directory for the file Program.dbo. If it is not found, dBASE IV will search the C:\DBASE\DBDATA subdirectory.

```
. SET PATH TO C:\DBASE\DBDATA
. DO Program
```

See Also

DIR, SET(), SET DEFAULT, SET DIRECTORY

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

SET()

See Chapter 6, ''SQL Commands'' for more information on SQL commands.

SET POINT changes the character used for the decimal separator.

Syntax

SET POINT TO [<expC>]

Default

The default for SET POINT is a period (.).

Usage

Use this command to change the decimal separator from the period (.) to the comma (,) for international usage. You can also specify another quoted single character for the decimal separator; but not a letter, a digit, or a space.

SET POINT only affects the screen display and printed output. To allow dBASE IV programs portability in international markets, dBASE IV uses the period as the decimal separator for internal representations, regardless of the SET POINT setting. Therefore, when entering a literal value, you must use a period as the decimal separator.

To restore the default, which is a period (.), type SET POINT TO ↵.

Example

To enter a literal number, then display it with a comma decimal separator:

```
. SET POINT TO ","
. x = 99.99
      99,99
. ? x
      99,99
```

See Also

SET SEPARATOR

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>]

Default

The default for SET PRECISION IS 16. The range of the optional numeric expression can be from 10 to 20.

Usage

Type N numbers have a numeric accuracy of 15.9 digits. If you SET PRECISION TO 15 or less, dBASE IV uses this number in math operations, and all digits are accurate. If you provide a number from 16 to 20, digits beyond 15 may not be accurate.

Remember that operations involving type N and type F numbers will yield a type F result. Type F numbers have a precision of 15.9, regardless of the SET PRECISION value.

See *Getting Started* for information on changing the default PRECISION value in the Config.db file.

See Also

FIXED(), SET(), 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.

SET PRINTER

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 device PRN. You can assign the printer to any one of the three parallel ports (LPT1, LPT2, LPT3) or one of four serial ports (COM1, COM2, COM3, or COM4). You can also SET PRINTER TO NUL to provide a nonexistent device to test applications. With SET PRINTER TO NUL, the write operations are simulated, but no data is actually written. For example:

```
. SET PRINTER TO LPT2
```

sends print output to the printer connected to parallel port 2.

If the message **Printer not ready** appears when you try to access a serial printer, even though the printer is on-line, you can try redirecting the COM port to an LPT device, then SET PRINTER to the LPT device. For example, use MODE LPT3 = COM1 from DOS, then in dBASE IV, type:

```
. SET PRINTER TO LPT3
```

Printing on a Network

In a network environment, dBASE IV signals the file server to execute the most recent print job. 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.

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 Employee 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, PRINTJOB/END PRINTJOB, REPORT FORM, SET(), SET DEBUG, SET DEVICE, _pdriver

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. If dBASE IV cannot find a .dbo file, it will look for and compile a program (.prg) 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 963 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.prg - Procedures

PROCEDURE Proc1
   ? "This is a message from Proc1 of Proctest.prg."
RETURN
* EOP: Proc1

PROCEDURE Proc2
   ? "This message is in Proc2 of Proctest.prg."
   DO Proc3
RETURN
* EOP: Proc2

PROCEDURE Proc3
   ? "This message is in Proc3 of Proctest.prg and"
   ? " was called from Proc2 of Proctest.prg."
RETURN
* EOP: Proc3
* EOP: Proctest.prg
```

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), type

```
. DO Setproc
This is a message from Proc1 of Proctest.prg.
This message is in Proc2 of Proctest.prg.
This message is in Proc3 of Proctest.prg and
 was called from Proc2 of Proctest.prg.
```

See Also

COMPILE, DO, PARAMETERS, SET()

SET REFRESH

SET REFRESH determines how often dBASE IV refreshes the workstation screen with database information from the server.

Syntax

SET REFRESH TO <expN>

Default

The default is 0, which means the display is not updated.

Usage

SET REFRESH works only in a local area network (LAN) environment, and only on database files that have been CONVERTed.

The value of <expN> is a time interval expressed in seconds from 0 to 3,600.

If you SET REFRESH to an interval greater than 0, and then use BROWSE, EDIT, or CHANGE to edit records, the workstation screen is periodically updated with any changes written to the server by any workstation.

The refresh count is in effect until you QUIT or reset the count with another SET REFRESH command.

In BROWSE, all records in the current BROWSE table are reread from the server and redisplayed on the workstation screen. In EDIT and CHANGE, only the current record is reread from the server, as these commands present only one record at a time.

If any workstation has a lock on the file or records in the current display, the screen cannot be refreshed. Once the record or file is unlocked, the screen will be updated on the next periodic refresh check. The screen will not refresh while an EDIT or BROWSE menu is displayed.

Special Case

You may also set the refresh count in Config.db with the REFRESH setting.

SET REFRESH

Example

To refresh the screen from the server every five seconds during a BROWSE:

```
. USE Employee
. CONVERT
. SET REFRESH TO 5
. BROWSE
```

See Also

BROWSE, CONVERT, EDIT, SET()

SET RELATION

SET RELATION links the active database file to one or more open database files according to a common key expression.

Syntax

SET RELATION TO [<expression> INTO <alias>
[, <expresssion> INTO <alias>...]]

Default

SET RELATION TO ↵, without any additional parameters, removes the relation from the currently selected work area.

Usage

SET RELATION links the active database file with each database file identified by INTO <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*.

You cannot perform circular relations. For example, you cannot link database Test1 to database Test2 and then link Test2 back to Test1.

The relating expression may be any expression derived from the parent file data matching the key of the index controlling the child file. If the child file is not indexed, the relating expression must be numeric and evaluate to the record number of the target record in the child file.

If you use a numeric expression, the active database is always linked to the record number in the file specified by the numeric expression (typically 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.

With a SET SKIP list active, the record pointer is advanced in each database file, starting with the last work area in the relation chain and moving up the chain towards the parent file.

SET RELATION

Tips

You can save a working environment, including relations, with CREATE VIEW <.vue filename> FROM ENVIRONMENT, which creates a dBASE III PLUS .vue file. Remember that CREATE/MODIFY QUERY/VIEW is available to you as a menu-driven command for linking files. It also allows you to save relations to a .qbe (dBASE IV) 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 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

. 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:

```
. 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

. 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

. 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(), SET FIELDS, SET FILTER, 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(), ON ERROR, RLOCK()/LOCK(), SET(), 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

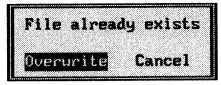
SET SAFETY ON/off

Default

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:



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.

The MODIFY COMMAND/FILE command and MODIFY STRUCTURE command save their original files as backup files before executing. These commands ignore SET SAFETY when creating the backup files, so old backup files are always overwritten.

If you ZAP a database file when SET SAFETY is ON, you will be asked to verify that you want to remove all records from this file.

See Also

COPY, COPY FILE, INDEX, JOIN, SAVE, SET(), SORT, TOTAL, UPDATE, ZAP

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. In a local area network (LAN) environment, **RecLock** and **FileLock** indicate a record or file lock.

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(), SET MESSAGE, SET STATUS

SET SEPARATOR

SET SEPARATOR changes the symbol used for the thousands 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 can be only one character long but cannot be a number or a letter. The default may be changed in the Config.db file. See *Getting Started*.

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 works with SET RELATION to link database files when there is a one-to-many or many-to-many relation between records in the parent and child files. It allows you to access all records in child files that match the relation key.

Syntax

SET SKIP TO [<alias> [,<alias>]...]

Usage

If a work area is included in the SKIP list, all child records matching the relation key in the child work area are processed, not just the first matching record. The order of the work areas in the list does not affect processing, but only work areas linked with SET RELATION should be included in the list. SET SKIP affects only the child work areas in the relation, so it isn't necessary to include the current work area in the SKIP list.

SET SKIP affects any command that moves the record pointer, such as BROWSE, SKIP, and all commands that allow FOR and WHILE clauses.

When a SKIP list is defined, dBASE IV advances the record pointer in the last work area in the relation chain, as long as this work area also appears in the SKIP list. For example, if the current work area is A, and you include only one work area in the SKIP list, work area B, with the command

SET SKIP TO B

a LIST command will first move the record pointer in B. When the record pointer in B reaches a record whose relation key no longer matches the current record in A, the record pointer in A advances. Therefore, all records in B that have the same relation key as A are processed first, then the record pointer in A moves to the next record. SET SKIP works similarly if three or more files are related. For example, if three files are related, A to B and B to C,

SET SKIP TO B, C

or

SET SKIP TO C, B

can be issued from work area A. Either of these commands tells dBASE IV to process all records with a matching relation key in C first, until a record with a different key in C is reached. Then, all records in B are processed, until a differing key is reached. Then the record pointer in A moves to the next record.

With A, B, and C still linked in the same relation, you can also type

SET SKIP TO C

from work area A. After all records with a matching key in C are read, the record pointer is advanced in A. Not every file in the relation chain must be included in the SET SKIP list.

Examples

Use SET SKIP to display all matching records in related database files.

```
. SELECT 3
. USE Stock ORDER Order_id
Master index: ORDER_ID
. SELECT 2
. USE Transact ORDER Client_id
Master index: CLIENT_ID
. SET RELATION TO Order_id INTO Stock
. SELECT 1
. USE Client          && Master database file.
. SET RELATION TO Client_id INTO Transact
. SET SKIP TO Stock, 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

SET SPACE is ON by default, for compatibility with dBASE III PLUS.

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
```

See Also

SET()

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.

See Also

SET()

If you wish to start in SQL as the default, see *Getting Started* 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, BROWSE, 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.

SET STATUS also controls the display of *scoreboard* information, which includes the **Del**, **Ins**, **Caps**, **Num**, **FileLock**, and **RecLock** indicators. If SET STATUS is ON, scoreboard information appears on the status bar. IF SET STATUS is OFF and SET SCOREBOARD is ON, scoreboard information appears in the upper right of the screen. If both SET STATUS and SET SCOREBOARD are OFF, the scoreboard function does not appear on the screen at all.

When you issue commands that cause information on the status bar to change, the status bar is updated.

Tip

If SET STATUS is ON, be sure not to specify fields or text below line 21 (or line 41 in EGA43 mode) in a format file. If you use the lower area of the screen, your programs or data will overwrite the status bar and the message line.

See Also

SET(), 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

Default

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(), 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. The record counter that some commands display is also disabled.

Example

After you SET TALK OFF, a command response is not displayed.

```
. Mnum=12345
12345
. SET TALK OFF
. Mchar ="Hello"
.
```

See Also

SET(), SET DEBUG, SET ECHO, SET ODOMETER, 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

ASSIST, EDIT, REPLACE, SET(), 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 while SET ESCAPE is ON. 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(), SET DEVELOPMENT, SET ESCAPE

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 also controls how many characters can be entered in the type-ahead buffer with the KEYBOARD command.

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(), KEYBOARD, ON, SET(), 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

Default

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.

dBASE IV processes UNIQUE indexes only once. Therefore, a previously hidden key value is not automatically updated when it is changed. REINDEX explicitly updates all key values in a UNIQUE index.

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.

SET UNIQUE

Example

To LIST all of the Part_name 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(), 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(), 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

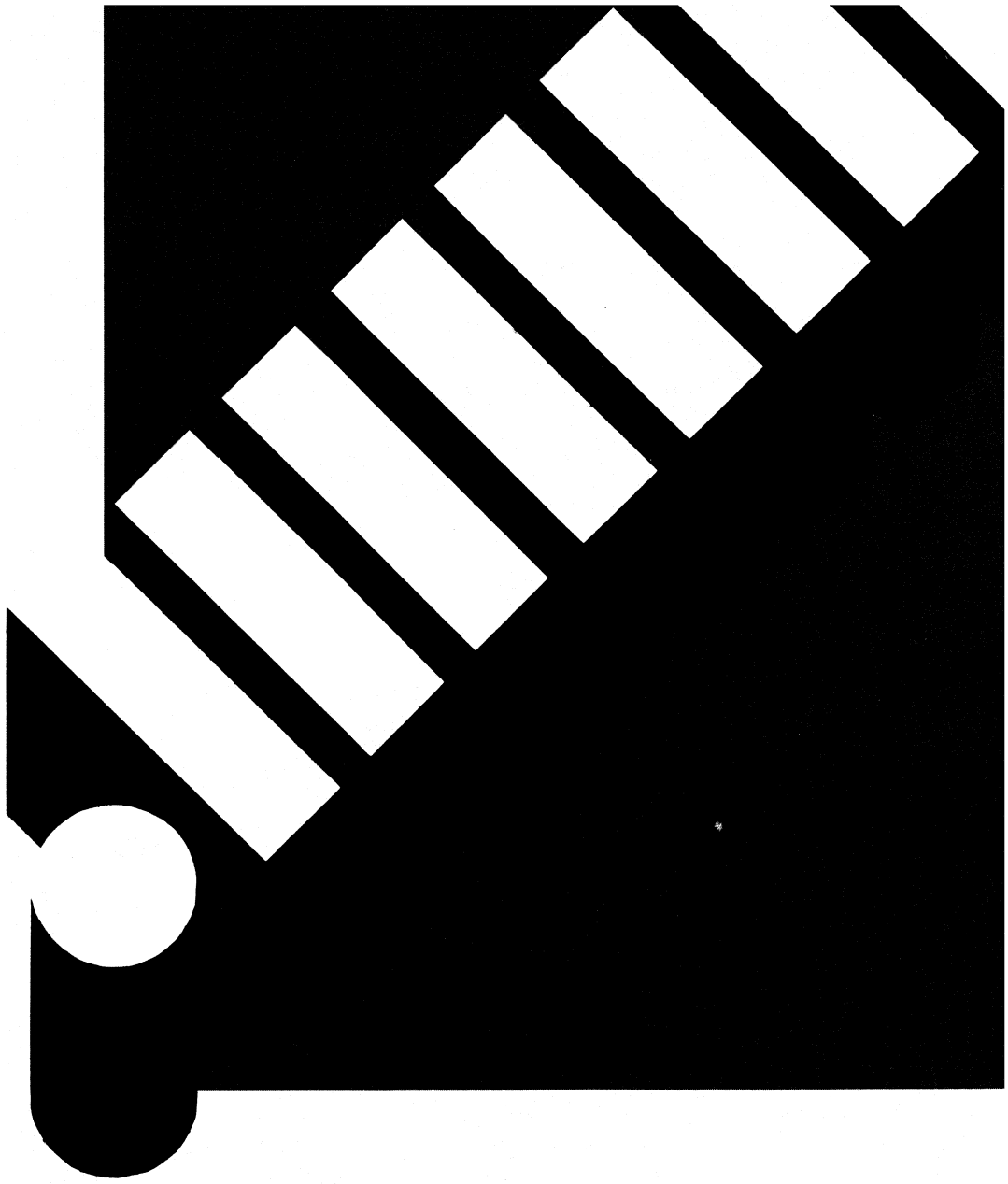
The first and the third @...GET commands use the window defined as John to edit memo fields. The second @...GET uses the window defined as Mary.

```
. 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 SET WINDOW OF MEMO TO to an undefined window name, the error message **WINDOW has not been defined** appears when you try to edit the memo field.

See Also

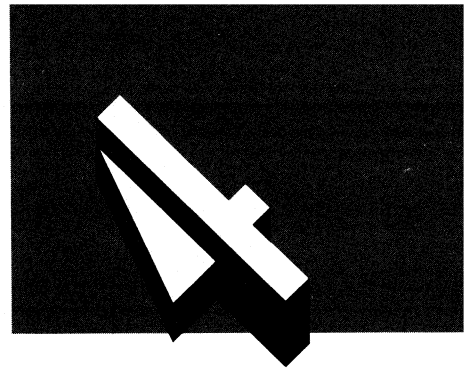
ACTIVATE WINDOW, CLEAR WINDOW, DEFINE WINDOW, MOVE WINDOW



i 1 2 3 4 5 6 7 A B

C D E F G H 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.

You cannot use structured programming commands in macros because dBASE IV cannot compile them at runtime. These include DO WHILE loops, DO CASE constructs, IF constructs, SCAN...ENDSCAN, TEXT...ENDTEXT, PRINTJOB...ENDPRINTJOB, and BEGIN/END TRANSACTION. You can, however, use macros in the condition portion of an IF, CASE, DO WHILE, or SCAN command.

You cannot use macro substitution on a database character field.

Because the macro substitution function is evaluated only once in a DO WHILE condition, you should only use the & macro if its value will not change within the DO WHILE loop.

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:INVOICES.DBF
```

A new method in dBASE IV uses indirect file references rather than macro substitution. This method does not call the compiler and provides fast execution:

```
. USE "C:"+Mdirectory+"\"+Filename
```

Refer to the Filenames section in Chapter 1 for complete information on indirect filename references.

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 = {12/25/90}
12/25/90
. date2 = {04/01/90}
04/01/90
. ? ABS(date1 - date2)
        268
. ? ABS(date2 - date1)
        268
```

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
```

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.

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.

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
. 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

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 numeric expression is the tangent of a particular angle. The value returned 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))
      0.50

. y=COS(DTOR(30))
      0.87

. ? RTOD(ATN2(x,y))
       30
```

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() = 1
       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 the record pointer is positioned before the first logical record of the specified file.

If you issue SKIP −1 when the record pointer is on the first record, BOF() is true (.T.), which indicates that the record pointer is at the beginning of the file. However, RECNO() remains at 1.

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(), RECNO(), SKIP

The CALL() function is used in programming to execute binary program modules.

Syntax

CALL(<expC>,<expression>,[<expression list>])

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 the value returned by the function in a dBASE IV expression.

Refer to the LOAD command in Chapter 2 for details on writing the binary program.

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.

CALL() accepts eight total arguments. The first value is always the module name, the remaining seven are parameters passed to the program. Since a function can return only one value, only the second value passed to CALL() is returned. The remaining variables are updated in memory, but are not returned by the CALL() function.

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

CALL, DECLARE, LOAD, RUN, RUN(), 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
. second = 10
         10
. ? CEILING(first / second)
          13
```

CEILING(), unlike ROUND(), always returns an integer closer to zero for negative numbers.

```
. ? CEILING(-5.556)
           -5
. ? ROUND(-5.556,0)
          -6
```

See Also

FLOOR(), INT(), ROUND()

CERROR()

The CERROR() function returns the number of the last compiler error message.

Syntax

CERROR()

Defaults

CERROR() returns a zero if no compiler errors are encountered.

Usage

CERROR() is updated each time the dBASE IV compiler executes. The compiler can be activated with the COMPILE command. It is also executed by DO, DEBUG, and SET PRODECURE; as a result of creating reports, forms, labels, queries, and applications; and when entering commands at the dot prompt.

When a compilation error occurs, ERROR() returns **360**, and MESSAGE() returns **Compilation error**. CERROR() returns the error number of the compiler error that occurred.

Error messages are listed in Appendix A of this manual. Potential returns from CERROR() are marked with an asterisk in this appendix. CERROR() is not affected by warning messages generated at compile time, such as the following:

Command not functional in dBASE IV	**[93]**
Extra characters ignored at end of command	**[151]**
Command not valid in RunTime environment	**[348]**
Command will never be reached	**[369]**

Before executing a new program, use the CERROR() function to test whether the source code compiled successfully.

Example

The following program segment uses CERROR() in a DO WHILE loop to force the user to edit the program until it compiles successfully.

```
DO WHILE .T.
   CLEAR
   MODIFY COMMAND User.prg
   ON ERROR ? ERROR(), MESSAGE()
   COMPILE User.prg
   ON ERROR
   IF CERROR() > 0
      ?
      WAIT "Your program did not compile. Press a key to fix it."
      LOOP
   ENDIF
   EXIT
ENDDO
DO User
```

See Also

COMPILE, DEBUG, DO, ERROR(), MESSAGE(), ON ERROR, SET PROCEDURE

CHANGE()

The CHANGE() function determines if another user has changed a record since it was read from the database file.

Syntax

CHANGE([<alias>])

Usage

CHANGE() can be used only with database files that have been updated with the CONVERT command. A CONVERTed database file has a field called __dbaselock which contains a counter that dBASE IV updates whenever the record changes.

The CHANGE() function works by comparing the counter in the workstation memory image of __dbaselock to the counter stored on disk. If they are different, CHANGE() returns a logical true (.T.). If the counters are the same, CHANGE() returns a logical false (.F.).

You can reset the value of CHANGE() 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.

You can include an alias to test a record in another work area.

NOTE
When a record is updated in a transaction, its counter is immediately updated. If you use the CHANGE() function at this point, it returns a true. However, if the transaction is not completed and a rollback occurs, the change to the record is no longer valid.

See Also

BEGIN/END TRANSACTION, COMPLETED(), CONVERT, FLOCK(), LKSYS(), RLOCK(), ROLLBACK, ROLLBACK(), SET REFRESH

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. When used with system memory variables _pscode and _pecode, the CHR() function should not be placed in quotation marks or other delimiters. For example,

_pscode = CHR(27)+"E"

is correct, but

_pscode = "CHR(27)+E"

is not.

You cannot use CHR() in print menus to send starting and ending 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.

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.). If you ROLLBACK before the END TRANSACTION, however, COMPLETED() remains false (.F.).

Example

To test for the successful completion of a transaction in a program file:

```
BEGIN TRANSACTION
      .
      .
      .
END TRANSACTION
IF .NOT. COMPLETED()
   IF .NOT. ROLLBACK()
      ? "The transaction was not completed and could not"
      ? "be restored. Recover manually."
      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>)
or
{mm/dd/yy}

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

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()
  04/01/90
```

To store the system date to the memory variable Mdate:

```
. STORE DATE() TO Mdate
  05/04/90
```

or:

```
. Mdate = DATE()
  02/29/91
```

To calculate the number of days between two dates:

```
. ? DATE()
    08/15/90
. Thirdq = {07/01/90}
    07/01/90
. ? "Day " + STR(DATE() - Thirdq,5,0)
    Day    45
```

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
. ? Mday
          15

. ? TYPE("Mday")
N
```

See Also

CDOW(), DATE(), DOW()

DBF()

The DBF() function returns the name of the database file in USE in the specified selected work area.

Syntax

DBF([<alias>])

Defaults

If no alias is specified, DBF() defaults to the current work area. 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(), SET FULLPATH, 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  Fred
      2  Sandra
      3  Ric
      4  Kimberly
      5  George J.
      6  Gene
      7  Florence
      8  Riener
. newname = "Kimbrelee"
Kimbrelee
. LIST Firstname FOR DIFFERENCE(Firstname, newname) > =2
Record#  Firstname
      2  Sandra
      4  Kimberly
      8  Riener
. 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
```

To store the day of the week from the system date to the memory variable Dayofweek:

```
. Dayofweek = DOW(DATE())
        6
. ? Dayofweek
        6
```

To determine the number of the day of the week for a date 197 days in the future:

```
. ? DOW(DATE() + 197)
        7
```

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(), DTOS(), 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
. 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. This ensures that the date is indexed properly when concatenated to character strings in an index expression.

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**.

If you issue a 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.).

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(), FIND, FOUND(), LOCATE, RECCOUNT(), RECNO(), SEEK, SEEK()

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

CERROR(), 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 uppercase 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
. Mfield = FIELD(number)
CLIENT
```

See Also

DBF(), LUPDATE(), RECCOUNT(), RECSIZE(), VARREAD()

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.

FILE() attempts to open the file briefly to determine if it exists.

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 DIRECTORY, SET PATH

FIXED()

The FIXED( ) function converts floating point numbers to Binary Coded Decimal numbers.

Syntax

FIXED(<expN>)

Usage

The FIXED( ) function is used to convert type F (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( ), SET PRECISION

FKLABEL()

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 *Getting Started.)*

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

FLOAT()

The FLOAT() function converts Binary Boded Decimal (type N) numbers to 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.

The range for <expN> is based on the setting of SET PRECISION TO <expN>. The largest number allowed is 0.9 × E + 308. The smallest number allowed is 0.1 × E − 307. The range for type F numbers is the same.

Any expression containing a type F number returns a type F result.

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
. ? FLOOR(first / 10)
        12
```

See Also

CEILING(), INT(), MOD(), ROUND()

FOUND()

The FOUND() function returns a logical true (.T.) if the previous FIND, LOCATE, SEEK, or CONTINUE command in the specified work area 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.

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.

Syntax

GETENV(<expC>)

Usage

GETENV() enables you to check the settings of any of the DOS environment commands. If dBASE IV does not find the environment command parameter, it returns a null string.

The name of the system variable may be entered as a character string, a character variable, or a combination of both. 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. The IIF() function does not evaluate <exp 1> and <exp 2>, it just returns them when the condition is true or false.

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. This is useful when creating queries, reports, labels, and indexes.

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 field 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.This numeric argument may include fractions of a second. INKEY(0) causes INKEY( ) to wait indefinitely.

INKEY( ) (without a parameter) 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. INKEY( ) does not return the value for **Ctrl**-S or the left arrow (←) key (decimal 19) unless SET ESCAPE is OFF.

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 uppercase and lowercase letters. The values returned are decimal.

Table 4-1 INKEY() returned values

Key Names		Decimal value
Ctrl-A	**Ctrl**-←	1
Ctrl-B	**End**	2
Ctrl-C	**PgDn**	3
Ctrl-D	→	4
Ctrl-E	↑	5
Ctrl-F	**Ctrl**-→	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	**Home**	26

(*continued*)

Table 4-1 INKEY() returned values (*continued*)

Key Names		Decimal value
Esc	**Ctrl-[**	27
F1	**Ctrl-**	28
Ctrl-]	**Ctrl-Home**	29
Ctrl-^	**Ctrl-PgDn**	30
Ctrl-PgUp		31
Spacebar		32
Backspace		127
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

(*continued*)

Table 4-1 INKEY( ) returned values (*continued*)

Key Names	Decimal value
Ctrl-F8	−17
Ctrl-F9	−18
Ctrl-F10	−19
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

Table 4-2 **Alt** key combination INKEY() values

With Letters			With Numbers
a −435	k −425	u −415	1 −451
b −434	l −424	v −414	2 −450
c −433	m −423	w −413	3 −449
d −432	n −422	x −412	4 −448
e −431	o −421	y −411	5 −447
f −430	p −420	z −410	6 −446
g −429	q −419		7 −445
h −428	r −418		8 −444
i −427	s −417		9 −443
j −426	t −416		0 −452

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)
   IF i <= 0
      i = 0
   ENDIF
   DO CASE
         CASE CHR(i) $ "Qq0"
              RETURN
         CASE CHR(i) $ "Aa1"
              DO <procedure 1>
         CASE CHR(i) $ "Bb2"
               DO <procedure 2>
   ENDCASE
ENDDO
```

See Also

CHR(), KEYBOARD, LASTKEY(), ON KEY, READ, READKEY(), SET TYPEAHEAD

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 uppercase or lowercase 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.

ISCOLOR() returns a .T. if a monchrome monitor is run by a color card, as in COMPAQ portables. The display may be in shades of grey instead of color.

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.

See Also

SET COLOR, SET DISPLAY

ISLOWER()

The ISLOWER() function returns a logical true (.T.) if the specified character string begins with a lowercase alphabetical character.

Syntax

ISLOWER(<expC>)

Usage

Use this function to evaluate and manipulate character strings.

ISLOWER() returns a logical false (.F.) when either an uppercase letter or a non-alphabetical character is in the first position of the expression.

Example

To test whether an expression begins with a lowercase alphabetical character:

```
. ? ISLOWER("abc123")
.T.
. ? ISLOWER("ABC123")
.F.
. ? ISLOWER("12abc")
.F.
```

See Also

ISALPHA(), ISUPPER(), LOWER(), UPPER()

ISMARKED()

ISMARKED() returns a logical true (.T.) if a change has been made to a database file during a transaction.

Syntax

ISMARKED([<alias>])

Usage

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.

Use ISMARKED() after a BEGIN TRANSACTION to establish whether the database file is in an integral state or 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(), RESET, ROLLBACK, ROLLBACK()

ISUPPER()

The ISUPPER() function returns a logical true (.T.) if the specified character expression begins with an uppercase alphabetical character.

Syntax

ISUPPER(<expC>)

Usage

Use this function to evaluate and manipulate character strings.

ISUPPER() returns a logical false (.F.) when either a lowercase letter or a non-alphabetical character is in the first position of the string.

Example

To test whether a string begins with an uppercase 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([<expC>,] <expN> [,<alias>])

Usage

The character expression is an .mdx filename associated with the current database file. 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>. The .ndx files appear prior to the .mdx tags in this instance.

The order of tags returned is:

- .ndx file expressions (if open)
- production .mdx tags
- non-production .mdx tags

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 to exit a full-screen command.

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>/<memo field 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 or memo field. 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. LEFT() returns a maximum of 254 characters from any string or memo field.

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

LTRIM(), RTRIM()

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, or no characters at all, 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* is "abc" or if it ends with the characters "abc". Similarly, LIKE("*abc*",astring) will return a true value if *astring* contains the characters "abc" anywhere in it.

This function is case sensitive: "ABC" and "abc" will not match. Wildcard characters, however, may stand for uppercase or lowercase letters.

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.

. ? LIKE("?BARBARA","BARBARA")
.F.

. ? LIKE("*BARBARA","BARBARA")
.T.
```

LINENO()

LINENO() returns the line number of the command 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.

Example

The following program file contains a procedure that uses LINENO() to return the line number causing the error.

```
ON ERROR DO Err_proc WITH LINENO(), PROGRAM()
.
.
.
PROCEDURE Err_proc
PARAMETERS err_line, prog_name
? "An error occurred on line number "+LTRIM(STR(Err_line))
? "of the program "+TRIM(prog_name)
? "Write down the line number and program name,"
? "and call the programmer with this information."
ON ERROR
RETURN
```

See Also

DEBUG, PROGRAM(), SET DEVELOPMENT

LKSYS()

The LKSYS() function determines the log-in name of the user who has locked a record or a file that you have tried unsuccessfully to access, and the date and time of the lock. It also returns the time, date, and log-in name of the user who last updated the record or file.

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 or file.
n = 3 returns the time of the last update or lock.
n = 4 returns the date of the last update or lock.
n = 5 returns the log-in name of the user who last updated or locked the record or file.

Usage

Arguments 0, 1, and 2 only return values after an attempted file or record lock has failed. That is, the file or record is currently locked by someone else, and argument 2 returns the name of the user who holds the current lock. Arguments 3, 4, and 5 may be used whether or not the record or file is currently locked.

This function will return a null string, unless the CONVERT command has been used to add the __dbaselock field to the active database file. 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 for arguments 2 and 5. To get the full user log-in name, use an argument of 24 with CONVERT.

Whenever a record is locked, the time, date, and user log-in name are placed into the __dbaselock field in the record. When the database file is locked, this same information is stored in the __dbaselock field of the first physical record in the file. The updates occur for both automatic and explicit file and record locks.

The first three arguments return information about the last failed lock attempt. If a file or record lock on a converted database file fails, the information for arguments 0, 1, and 2 is written to a buffer from the database's __dbaselock field. If you use LKSYS() with arguments 0, 1, or 2, the time of lock, date of lock, or name of user holding the lock is read from this buffer. This buffer is not rewritten until you attempt another lock that fails. Therefore, 0, 1, and 2 always return the information that was current at the time of the last lock failure.

The last three arguments return information about the last successful record or file lock. Arguments 3, 4, and 5 return information directly from the __dbaselock field, not from an internal buffer.

LKSYS()

Examples

The following program example attempts to lock the current record. If the lock is unsuccessful, a message tells when the record was locked and by whom. Given this information, the user can decide whether to retry the lock.

```
DO WHILE .T.
   IF .NOT. RLOCK()
      ? "Record locked on " + LKSYS(1) + ;
      " at " + LKSYS(0) + " by " + LKSYS(2)
      WAIT "Do you want to try again?" TO mretry
      IF UPPER(mretry) = "N"
         RETURN
      ENDIF
   ENDIF
   EXIT
ENDDO
```

The following example shows how to determine the date and time a record was last locked.

```
? "Record last locked on " + LKSYS(4) + " at " + LKSYS(3)
```

See Also

CHANGE(), CONVERT, FLOCK(), RLOCK(), SET LOCK, UNLOCK

LOCK()

LOCK() is an alternate form of the RLOCK() function. See the RLOCK() function entry in this chapter.

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 y 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 x, 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
. y = 2
            2
. ? 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 field in a specified database file.

Syntax

LOOKUP (<return field>, <look-for exp>, <look-in field>)

Usage

This function searches <look-in field> for the <look-for exp>. If it finds a record that contains the expression, it returns the <return field> value.

You must specify an alias when referring to fields outside the current work area.

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 to the first record of the database file searched 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. LOOKUP( ) returns the value found in <return field>, which can be used in an expression.

Example

To retrieve information from a database file not in the current work area:

In this example, the Cust database contains customer names, a part number for a recent order, the quantity of items ordered, and a discount to be applied on this order.

FIND points to a record in the Cust database matching a given customer number. The customer is Johnson, who ordered part 645.

The LOOKUP( ) function in the next command line searches for part number 645 in the Partno field of the Parts database file. If it finds a match, it returns the value contained in the Cost field. The rest of the command line computes an amount using Cost (from the Parts database file); and Quantity and Discount (from the Cust database file).

```
. SELECT 1
. USE Parts
. SELECT 2
. USE Cust ORDER TAG Custno
. FIND 1130
. ? Lname, Pno
Johnson         645
. pline2 = LOOKUP(Parts->Cost, Pno, Parts->Partno) * Quantity - (Quantity * Discount)
```

LOWER()

The LOWER() function converts uppercase letters in a character string to lowercase letters. It does not work with memo fields unless the MLINE() function is used first to extract a character string from the memo field.

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 lowercase:

```
. charstring = "UPPERCASE"
  UPPERCASE
. newstring = LEFT(charstring,1) + SUBSTR(LOWER(charstring),2)
  Uppercase
```

See Also

ISALPHA(), ISLOWER(), ISUPPER(), MLINE(), 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()

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(<expression1>,<expression2>)

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(), SET FULLPATH, 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

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()
or
MEMORY(0)

Usage

MEMORY(0) and MEMORY() are alternate forms of this function. 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.

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

CERROR( ), ERROR( ), ON ERROR

The MIN() function returns the smaller value of two numeric, date, or character expressions.

Syntax

MIN(<expression1>,<expression2>)

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
```

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.

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(), MDX(), ORDER(), RECCOUNT(), RECSIZE(), SET FULLPATH, 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 uppercase. If there is no .ndx file, then it returns the .mdx tagname, also in uppercase.

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 INDEX, SET ORDER, TAG(), USE

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(), NETWORK(), 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.

The choice stored by PAD() is not cleared by DEACTIVATE MENU or DEACTIVATE POPUP.

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. If you are calculating a monthly payment using an annual interest rate, divide the rate by 12.

<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()      && PCOL() returns 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
. 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.

If no pop-up menu has been defined, POPUP( ) 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.

WARNING
Only directly hard-wired printers will return an accurate PRINTSTATUS(). Network printers or those connected by an intervening device may not return an accurate response.

Example

The following code checks the status of 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, procedure, or user-defined function that was executing when an error occurred.

Syntax

PROGRAM()

Usage

This function returns the name, as a character string, of the program file, procedure, or user-defined function 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. You can also use PROGRAM() with ON ERROR to identify the program that was executing when the error occurred. 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 " + LTRIM(STR(LINENO()))
```

See Also

DEBUG, LINENO(), ON ERROR, 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 are no popups or menus in memory, this function returns a null string. Deactivated popups and menus still return PROMPT() values.

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 uppercase
STRUCTURE — Returns the field name in uppercase

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.

PROW() does not address row and column positions within a print file.

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>])

Usage

The RAND() function computes a random number with or without a numeric argument. To get subsequent random numbers in the series, repeat the function with no parameters.

The numeric expression is 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. The expression is truncated to an integer, so RAND(1.1) and RAND (1) have the same value. RAND() and RAND(0) are equivalent.

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
2	258	**Ctrl**-A, **Ctrl**-←	beginning of previous word
3	259	**Ctrl**-F, **Ctrl**-→	beginning of next word
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

(continued)

Table 4-3 READKEY() coding (*continued*)

Non-Updated Code Number	Updated Code Number	Key Pressed	Keypress Meaning
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()
        1
. GO BOTTOM
  CLIENT: Record No 8
. ? RECNO()
        8
. SKIP
  CLIENT: Record No 9
. SELECT 2
. ? RECNO(1)
        9
. ? RECNO(Client)
        9
```

See Also

SET RELATION

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> + 34.

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 memo field, starting from the last character on the right.

Syntax

RIGHT(<expC>/<memo field name>, <expN>)

Usage

The RIGHT() function allows you to retrieve the last part of a character string or a memo field. The numeric expression defines the number of characters to extract from the character string or memo field.

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. RIGHT() returns a maximum of 254 characters from any string or memo field.

Example

To extract the last three characters from a character expression:

```
. ? RIGHT("abcdef",3)
def
```

See Also

AT(), LEFT(), LTRIM(), RTRIM(), STUFF(), SUBSTR()

RLOCK()

The RLOCK() function is used to lock multiple records. This function is identical to the LOCK() function.

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_name 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

Use ROLLBACK() to test for the successful completion of a ROLLBACK command.

This function returns true (.T.) by default. If a ROLLBACK fails for any reason (for example, a disk error), ROLLBACK() is 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)
          17.321111

. ? 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. When there is no active window and ROW() is tested from the dot prompt, it returns 21 with SET STATUS ON, and 24 with SET STATUS OFF. In 43-line mode, ROW() returns 39 with SET STATUS ON, and 42 with SET STATUS OFF.

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. If SET SPACE is ON, TRIM(<field name>,<field name>) inserts one blank space between the two fields. TRIM(<field name>) + <field name> does not insert a blank space between the fields, whether SET SPACE is ON or OFF.

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(), SET SPACE

RUN()

The RUN() function returns a completion code after executing an operating system command or program from within dBASE IV.

Syntax

RUN(<expC>)

where <expC> is the name of a command, a batch file, or an executable program to be executed by the operating system.

Usage

RUN(<expC>) executes the command or program specified by <expC>. You must include the file extension (if applicable) and the correct drive letter and path (if the program is not on the default drive and path). You can place command line arguments after the command or program name.

dBASE IV uses the DOS COMSPEC environment variable to locate the command interpreter. The file that COMSPEC specifies must exist to use RUN(). If dBASE IV cannot find the file, the message **Cannot load Command.com** appears.

Your computer must have enough available memory to load the command interpreter and the program that it will execute. dBASE IV frees up some memory before it executes the external program.

Do not execute programs that terminate and stay resident in memory (TSRs). Some DOS programs, such as PRINT and ASSIGN, are TSRs. You should run these commands before loading dBASE IV.

RUN() returns a numeric argument, but this argument is currently not defined.

Example

This example executes the Prog1.exe program:

```
. RUN("D:\Progs\Prog1.exe")
```

See Also

CALL, CALL(), LOAD, RUN/!

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.

If SET NEAR is ON, the record pointer moves to the record immediately following the sought-after expression whenever an exact match cannot be 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
L00001
. ? SEEK(Client_id, "Client")
.F.
. ? Client->Client_id

.
```

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

EOF(), FIND, FOUND(), LOOKUP(), SEEK, SET NEAR

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. If all ten work areas are used, SELECT() returns 0.

If a catalog is open and work areas 1 and 10 are used, SELECT() returns 9.

Example

```
. ? SELECT()
9
```

To open a file in an unselected area:

```
. Filename = "Client"
. USE (Filename) IN SELECT()
```

See Also

ALIAS()

The SET() function returns the status of various SET commmands.

Syntax

SET(<expC>)

Usage

Table 4-4 lists the keywords that are valid in the character expression <expC> and the values that the SET() function returns.

Table 4-4 SET() keywords and return values

SET Keyword	Value Returned	SET Keyword	Value Returned
ALTERNATE	ON/OFF	DESIGN	ON/OFF
ATTRIBUTES	SET COLOR values[1]	DEVELOPMENT	ON/OFF
		DIRECTORY	<path>
AUTOSAVE	ON/OFF	DISPLAY	<display mode>
BELL	ON/OFF	ECHO	ON/OFF
BLOCKSIZE	Integer	ENCRYPTION	ON/OFF
CARRY	ON/OFF	ESCAPE	ON/OFF
CATALOG	ON/OFF	EXACT	ON/OFF
CENTURY	ON/OFF	EXCLUSIVE	ON/OFF
CLOCK	ON/OFF	FIELDS	ON/OFF
COLOR	ON/OFF	FILTER	<filter expression>
CONFIRM	ON/OFF	FORMAT	<format filename>
CONSOLE	ON/OFF	FULLPATH	ON/OFF
CURSOR	ON/OFF	HEADINGS	ON/OFF
DEBUG	ON/OFF	HELP	ON/OFF
DECIMALS	Integer	HISTORY	ON/OFF
DEFAULT	<drive>	HOURS	12 or 24
DELETED	ON/OFF	INSTRUCT	ON/OFF
DELIMITERS	ON/OFF	INTENSITY	ON/OFF

(*continued*)

Table 4-4 SET() keywords and return values (*continued*)

SET Keyword	Value Returned	SET Keyword	Value Returned
LOCK	ON/OFF	REPROCESS	Integer
MARGIN	Integer	SAFETY	ON/OFF
MEMOWIDTH	Integer	SCOREBOARD	ON/OFF
NEAR	ON/OFF	SPACE	ON/OFF
ODOMETER	Integer	SQL	ON/OFF
PATH	<path>	STATUS	ON/OFF
PAUSE	ON/OFF	STEP	ON/OFF
PRECISION	Integer	TALK	ON/OFF
PRINTER	ON/OFF	TITLE	ON/OFF
PROCEDURE	<filename>	TRAP	ON/OFF
REFRESH	Integer	TYPEAHEAD	Integer
RELATION	<expression> INTO <alias>...	UNIQUE	ON/OFF
		VIEW	<filename>

[1]SET("ATTRIBUTES") returns a string of attributes as set with the SET COLOR command. The attributes returned are NORMAL, HIGHLIGHT, perimeter, then two ampersands (&&), then MESSAGES, TITLES, BOX, INFORMATION, and FIELDS.

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.
```

To determine the current settings of SET COLOR attributes:

```
. ? SET("ATTRIBUTES")
W+/B,RG+/GB,N/N && W/N,W/B,RG+/GB,B/W,N/GB
```

This indicates that NORMAL is bright white on blue, HIGHLIGHT is yellow on cyan, PERIMETER is black, MESSAGES are white on black (but may show up as bright white on black, see Table 3-4 under SET COLOR), TITLES are white on blue (but may also show up as bright white on blue), BOX is yellow on cyan, INFORMATION is blue on white, and FIELDS are black on cyan.

See Also

LIST/DISPLAY STATUS, SET

Chapter 3, "SET Commands"

SIGN()

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()

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.

```
. ? SIN(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

SPACE()

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.

The maximum value for the <length> option is 20 digits, and the maximum value for the <decimal> option is 18. If you supply a <length> or <decimal> option with a number greater than the maximum, an error message appears indicating that the value is out of range.

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

<expC1> 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 string without reconstructing the entire string. <expC2> is inserted into <expC1>, replacing <expN2> characters beginning with the character in position <expN1>.

If the string starting position indicated by <expN1> is zero, STUFF() treats it as 1. If it exceeds the length of <expC1>, it concatenates <expC2> to the end of <expC1>.

<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 <expC1>.

If <expC2> is a null string, dBASE removes the number of characters specified by <expN2> from <expC1> without adding characters.

NOTE
The STUFF( ) function is not supported in memo fields.

Example

To replace all hyphens in Field_1 with spaces:

```
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 you have an .ndx file, a production .mdx file, and a non-production .mdx file open, the order of the tags is .ndx, production .mdx, and then non-production .mdx.

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 trigonometric 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()

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().

Since TIME() returns the operating system time, the system clock must be set correctly for accurate results. TIME() is not affected by SET HOURS TO [12/24]. It will always return the time in 24-hour format.

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(<expression>,<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 <expression>, 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

TRIM()

TRIM() is an alternate form of the RTRIM() function. See the RTRIM() function entry in this chapter.

TYPE()

The TYPE() function evaluates a character argument and returns a single uppercase character that indicates whether the argument contains character, numeric, logical, memo, date, float, or undefined data.

Syntax

TYPE(<expC>)

Usage

The TYPE() function always returns an uppercase letter.

The possible TYPE() values are:

Character = C
Numeric = N
Logical = L
Memo = M
Date = D
Float = F
Undefined = U

You can use TYPE() to test for the existence of a variable or field. An undefined result (U) means the variable or field cannot be found.

The TYPE() argument must be a character string enclosed in quotes. The charcter string in quotes may refer to a field name or a variable name, but it must be the name of a data item. TYPE() then returns the type of data contained in the named data item.

If you use a character string without quotes as the argument, dBASE IV treats it like macro substitution and attempts to expand the argument. Therefore, only use an argument without quote marks if the argument can be expanded to a character expression. Once expanded, TYPE() returns the type of data named by the character expression.

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
```

With TYPE(), you can test if an expression is valid, or if it will return a **Data type mismatch** error message. To evaluate the expression "test/100" and determine if the expression is valid:

```
. test = "A"
A
. ? TYPE("Test/100")
U
```

TYPE() returned a U since a character string cannot be divided by a number.

```
. x = 5
          5
. y = 7
          7
. ? TYPE("x*y/4")
N
```

TYPE() returned an N since the operation evaluates to a valid expression.

To test an expression for validity, compare the result of the TYPE() function to the uppercase letter representing the expression type:

```
. ? TYPE("(1+2) = 5") = "L"
.T.
```

UPPER()

The UPPER() function converts lowercase letters in a character string to uppercase letters. It does not work with memo fields unless the MLINE() function is used first to extract a character string from the memo field.

Syntax

UPPER(<expC>)

Examples

```
. STORE "This is a nice day" TO niceday
This is a nice day
```

To display this variable in uppercase:

```
. ? 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 uppercase:

```
. 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(), MLINE()

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.

VAL() only recognizes the dot (.) as the decimal separator, and does not recognize the comma (,) as either a decimal separator or a thousands separator.

Examples

The value of a non-numeric character sequence in VAL() is zero:

```
. ? VAL("ABC")
  0
. ? VAL("A=123")
  0
. ? VAL("123=A")
  123

. SET DECIMALS TO 1
. y = VAL("123.45")
  123.5
. ? STR(y,6,2)
  123.45
```

VAL() displays 123.5, but stores 123.45 to memory.

See Also

SET DECIMALS, STR()

VARREAD()

Use the VARREAD() function in a procedure called by an ON command or UDF while a field, memory variable, or array element is being edited. The VARREAD() function is used with @ and full-screen editing commands to return the name of a field, memory variable, or array element that is in the process of being edited.

Syntax

VARREAD()

Usage

VARREAD() returns the field, memory variable, or array element names in uppercase letters. The array element name includes its row and column coordinates. To compare a character string with a field name returned by VARREAD(), be sure the string contains all uppercase letters.

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 LABEL 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, BROWSE, EDIT, FIELD(), 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 1.1
```

See Also

OS()

WINDOW()

The WINDOW() function returns the name of the currently active window.

Syntax

WINDOW()

Usage

This function returns the name of the current window as a character string. If no window is active, WINDOW() returns a null string.

Although several windows can be present on screen, only one window can be active. If you activate a list of windows with ACTIVATE WINDOW, only the last window in that list is active.

Example

You can save the name of the current window and activate it again later:

```
. DEFINE WINDOW Entry1 FROM 2,4 TO 15,30 DOUBLE
. DEFINE WINDOW Entry2 FROM 5,5 TO 20,20 DOUBLE
. ACTIVATE WINDOW Entry1
. Wcurrent = WINDOW()
. ACTIVATE WINDOW Entry2
.
.
.
. ACTIVATE WINDOW &Wcurrent
```

See Also

ACTIVATE WINDOW, DEFINE WINDOW

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
```

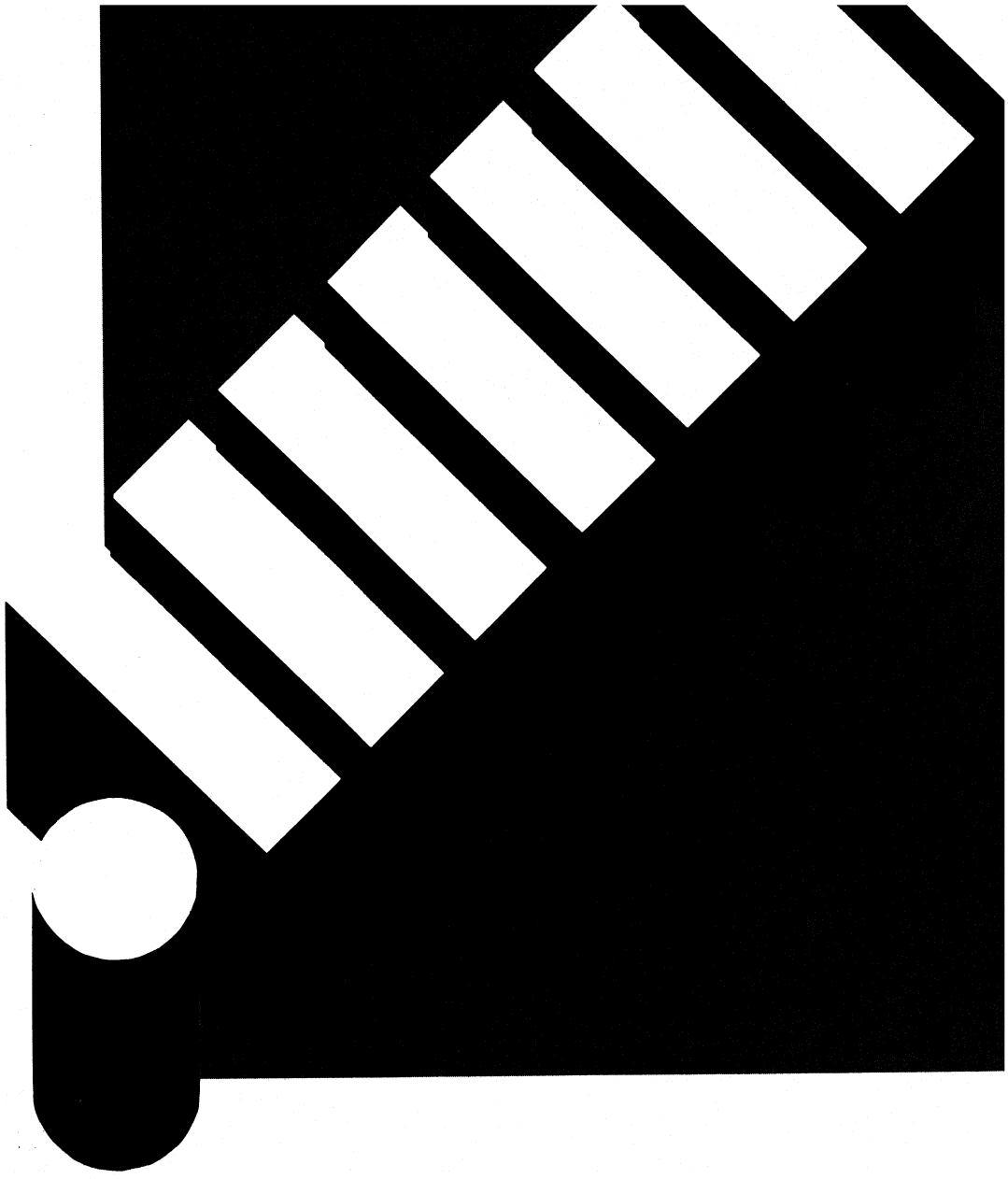
To store the year from the system date to the memory variable:

```
. 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:

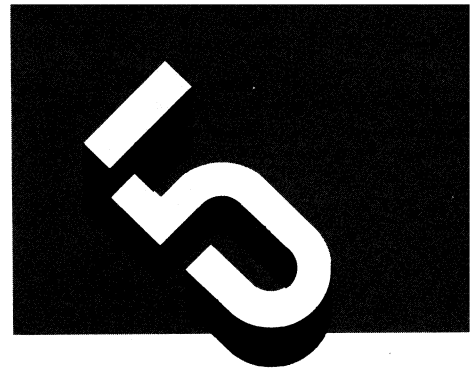
```
. Newyear = YEAR(DATE()) + 50
          2038
```



i 1 2 3 4 5 6 7 A B

C D E F G H In

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: 07/29/90
```

See Also

_lmargin, _rmargin, _wrap, @ (PICTURE functions B, I, and J), STR()

_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:

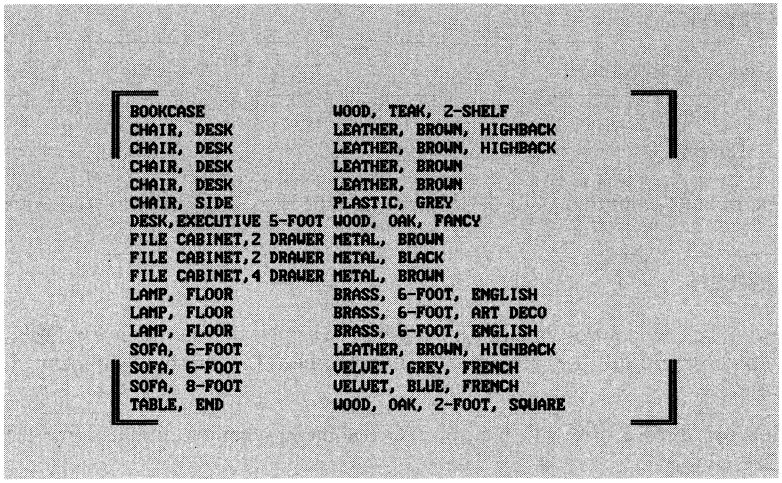


Figure 5-1 Boxed output

See Also

DEFINE BOX, SPACE()

_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 64.

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. The sum (_lmargin + _indent) must be less than the value of _rmargin. _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.

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 and _padvance = "LINEFEEDS", 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 out-put. 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

_pbpage and _pepage are relative to _pageno. These three system variables 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.

REPORT FORM and LABEL FORM always reset the page number to 1. To print a different page number on the first page of your report, access the **Print** menu and change the value of the **First page number** option.

Example

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
```

To print the second and third physical pages of a document, but number them page 5 and page 6:

```
_plineno = 0
_pageno = 5
_pbpage = 6       && second physical page
_pepage = 7       && third physical page
PRINTJOB
.
.
.
```

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.

Tips

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.

Remember that the current _pageno may or may not correspond to the current page being output, because _pageno is a system memory variable that can have an assigned value.

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. Assignment to _pcolno is equivalent to the AT clause of the ?/?? command.

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. You can assign _pcolno an integer value between 0 and 255.

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.

NOTE
Streaming output should not be mixed with @...SAY output.

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.

Example

The following program code segment overstrikes the word "altered" with slashes (/) 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 ////// 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.

NOTE
If _pcopies is greater than 1, the PRINTJOB command spools the output to your hard disk, and then sends it to the printer as many times as specified by _pcopies. Therefore, expressions in the PRINTJOB are evaluated only once. For example,

```
_pcopies=5
PRINTJOB
   ? "Report Copy",STR(x,1)
   x = x+1
   * Here's the rest of the report.
ENDPRINTJOB
```

creates five copies of "Report Copy 1".

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"
READ
_pcopies = copies
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 *Getting Started* 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.

The device that the driver prints to is the current default device (PRN or LPT1), which may not be the device you have configured in your Config.db file. Use SET PRINTER TO to change the device.

Usage

Printer drivers let you print styled text, such as **bold**, underline, 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.

Fonts configured in Config.db are associated by setting the driver with _pdriver statements.

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.

Output from the @ command is not sent through the printer driver.

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 border characters, with other printable 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.

The ASCII printer driver prints ASCII characters without interpreting them for the printer. If characters in a dBASE III PLUS report do not print correctly, but printed correctly in dBASE III PLUS, the printer driver may be interpreting these characters differently. Use the ASCII printer driver when printing this report.

Special Case

After activating the PostScript driver with _pdriver = "POSTSCRI.pr2", the first character you print downloads the PostScript .dld file to the printer. A few minutes after you stop printing, the printer may reset, thereby removing the .dld file from its memory. To be sure the .dld file is reloaded, issue another _pdriver = "POSTSCRI.pr2" command.

Example

This programming example asks the user 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

Getting Started

_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. You must include the curly braces { } around each code. 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
```

or

```
_pecode = "{27}E"             && Also sends Esc E
```

See Also

_pscode, ?/?? (STYLE option), ???, PRINTJOB/ENDPRINTJOB

Your printer manual has additional information about printer control codes.

_peject controls ejecting a sheet of paper before and 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 _peject at the dot prompt, 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

_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.

Example

You can optimize a custom report so that once _pepage is reached, no more records in the database file will be processed. The SCAN command line in the following procedure stops processing the database file as soon as the current page number (_pageno) is greater than the last _pepage. Without this optimization using the WHILE clause, this procedure would read every record in the Customer database file, even though you requested only the first three pages of the report to print.

```
USE Customer
_pageno = 1
_pepage = 3
_plength = 5
PRINTJOB
   SCAN WHILE _pageno <= _pepage
      ? Lastname
   ENDSCAN
ENDPRINTJOB
```

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, and the device, as contained in the PRINTER setting in Config.db.

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. You can also customize the settings in these print form files from any **Print** menu in the Control Center.When you **Save setting to print form**, the new settings are written to the .prf file.

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 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. _plineno is incremented whether output is directed to the screen or printer. Therefore, entering data on the screen or typing ↵ on the keyboard also affects the current line number.

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

_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

The default _ppitch is "DEFAULT", the pitch defined by your printer's DIP switch settings or setup code before you started dBASE IV. "DEFAULT" means that dBASE IV has not sent any pitch control codes to the printer.

If you change _ppitch to "PICA", "ELITE", or "CONDENSED", resetting it to "DEFAULT" has no effect. For example:

```
. ? _ppitch
DEFAULT
. _ppitch = "ELITE"
. ? _ppitch
ELITE
. _ppitch = "DEFAULT"
. ? _ppitch
ELITE
```

Since _ppitch = "DEFAULT" does not send any codes to the printer, the earlier setting, "ELITE", is still active.

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.)

"PICA" — 10 characters per inch.

"ELITE" — 12 characters per inch.

"CONDENSED" — approximately 17.16 characters per inch.

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. Remember that not all printers support both draft and letter-quality modes.

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

Appendix F, "dBASE IV Printer Drivers," lists the drivers available and modes supported.

_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. You must include the curly braces { } around each code. 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 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}"
```

_pscode

This sets Compressed Mode on an Epson FX+ printer:

```
_pscode = CHR(15)
```

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 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 79 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.

_rmargin

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, _wrap

Getting Started, 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.).

When _wrap is .T., streaming output is buffered until the current line is completed. If you output with a ? command, you may need to follow this with another ? command, as in the following example, to force the last line of text to print.

Example

_wrap determines whether or not the margin, alignment, and indent settings are activated:

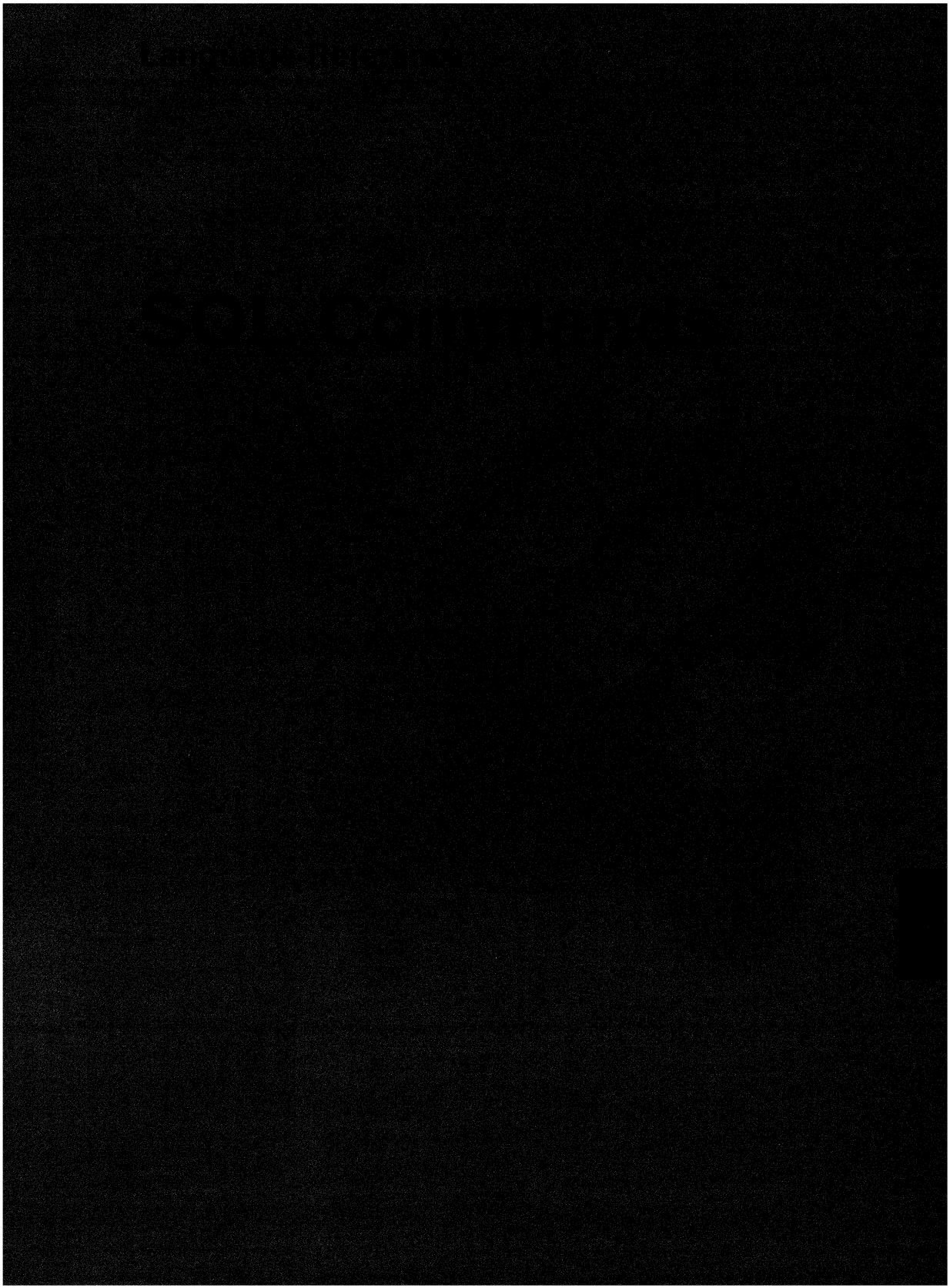
```
_indent = 3
_lmargin = 10
_rmargin = 30
Mem = "This is a sentence that contains almost sixty characters."
_wrap = .T.
? Mem
?
```

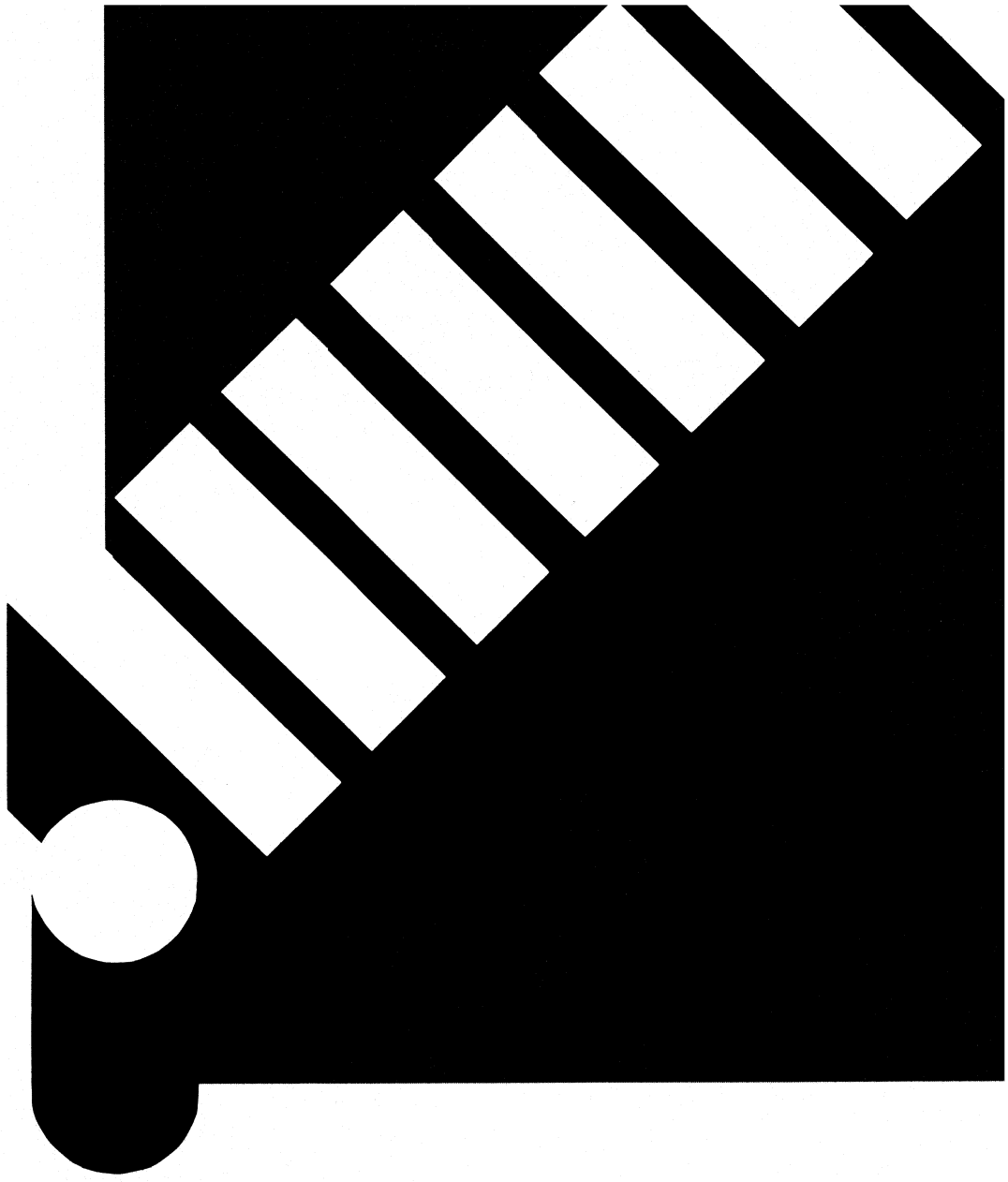
This code produces the following output:

```
          This is a
       sentence that
       contains almost
       sixty characters.
```

See Also

_alignment, _indent, _lmargin, _rmargin

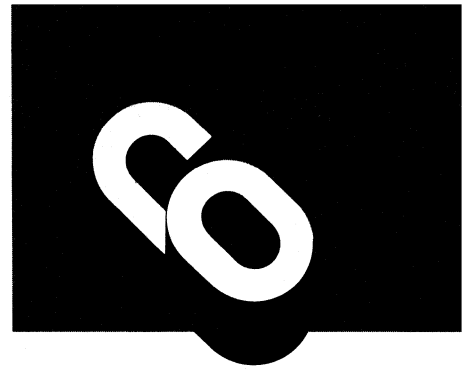




i 1 2 3 4 5 6 7 A B

C D E F G H In

SQL Commands



This section describes the syntax and usage of all SQL commands provided by dBASE IV. The commands are arranged in alphabetical order.

Symbols and Conventions

The syntax for SQL commands is provided in both written and graphic form. Figure 6-1 illustrates how to interpret each form of the command syntax.

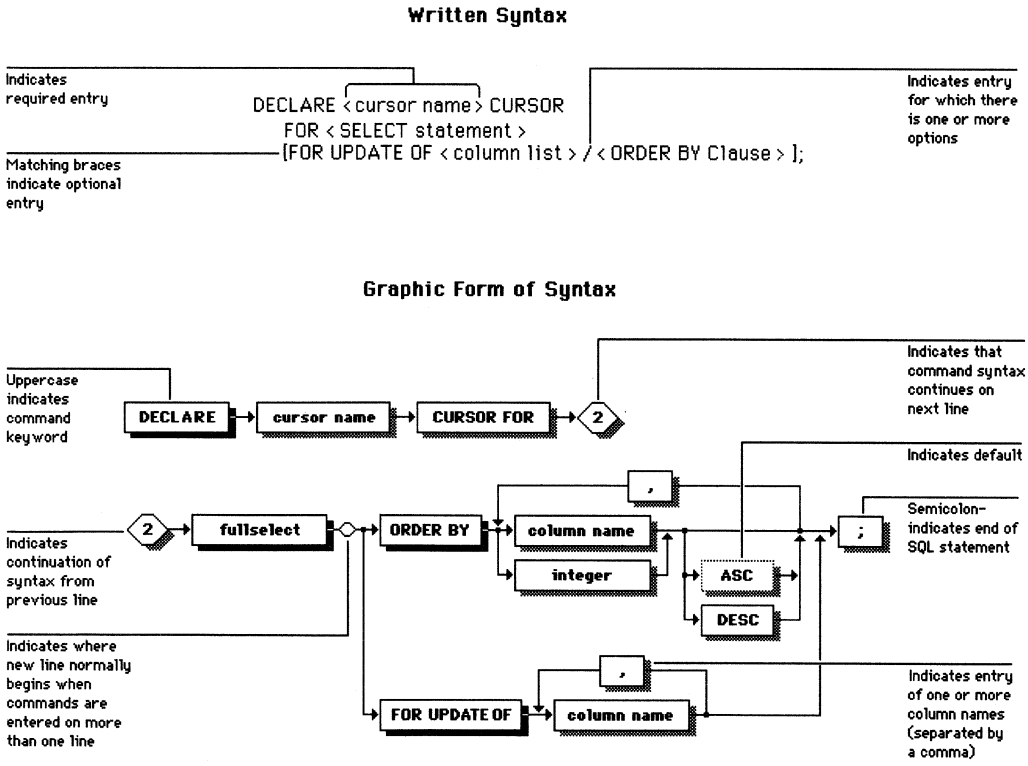


Figure 6-1 Command syntax

Reserved Words

The reserved words listed in Table 6-1 have special meanings and should not be used as names of tables, views, synonyms, indexes, columns, or memory variables.

Table 6-1 SQL reserved words

ABS	DECIMAL	LOG	SHOW
ACOS	DECLARE	LOG10	SIGN
ADD	DELETE	LOGICAL	SIN
ALL	DELIMITED	LOWER	SMALLINT
ALTER	DESC	LTRIM	SOUNDEX
AND	DIF	MAX	SPACE
ANY	DIFFERENCE	MDY	SQRT
AS	DISTINCT	MIN	START
ASC	DMY	MOD	STOP
ASIN	DOW	MONTH	STR
AT	DROP	NOT	STUFF
ATAN	DTOC	NUMERIC	SUBSTR
ATN2	DTOR	OF	SUM
AVG	DTOS	ON	SYLK
BETWEEN	EXISTS	OPEN	SYNONYM
BLANK	EXP	OPTION	TABLE
BY	FETCH	OR	TAN
CDOW	FIXED	ORDER	TEMP
CEILING	FLOAT	PAYMENT	TIME
CHAR	FLOOR	PI	TO
CHECK	FOR	PRIVILEGES	TRANSFORM
CHR	FROM	PUBLIC	TRIM
CLOSE	FV	PV	TYPE
CLUSTER	FW2	RAND	UNION
CMONTH	GRANT	REAL	UNIQUE
COS	GROUP	REPLICATE	UNLOAD
COUNT	HAVING	REVOKE	UPDATE
CREATE	IN	RIGHT	UPPER
CTOD	INDEX	ROLLBACK	USER
CURRENT	INSERT	ROUND	USING
CURSOR	INT	RPD	VAL
DATA	INTEGER	RTOD	VALUES
DATABASE	INTO	RTRIM	VIEW
DATE	KEEP	RUNSTATS	WHERE
DAY	LEFT	SAVE	WITH
DBASEII	LEN	SDF	WKS
DBCHECK	LIKE	SELECT	WORK
DBDEFINE	LOAD	SET	YEAR

NOTE
Do not use the names of dBASE commands and functions as names for SQL tables, views, synonyms, indexes, columns, or memory variables.

Classes of Commands

SQL commands in dBASE IV can be categorized by the type of operation they perform. The SQL commands are listed below by category. Notice that some commands are included in more than one category.

Creation/Startup of SQL Databases

CREATE DATABASE	Creates a database directory and a set of SQL catalog tables for the new database
SHOW DATABASE	Lists the currently available databases
START DATABASE	Opens (activates) a database
STOP DATABASE	Closes the current database

Creation/Modification of Objects

ALTER TABLE	Adds one or more new columns to an existing table
CREATE INDEX	Creates an index based on the values of one or more columns in a table or view
CREATE SYNONYM	Defines an alternate name for a table or view
CREATE TABLE	Creates a new table and defines its columns
CREATE VIEW	Creates a virtual table based on the columns in other tables or views

Database Security

GRANT	Grants user privileges for table access and update
REVOKE	Revokes GRANTed privileges

Data Definition

ALTER TABLE	Adds one or more new columns to an existing table
CREATE DATABASE	Creates a database directory and a set of SQL catalog tables for the new database
CREATE INDEX	Creates an index based on the values of one or more columns in a table or view
CREATE SYNONYM	Defines an alternate name for a table or view
CREATE TABLE	Creates a new table and defines its columns
CREATE VIEW	Creates a virtual table based on the columns in other tables or views
DBDEFINE	Creates SQL catalog table entries for database files defined as SQL tables
DROP DATABASE	Deletes a SQL database and all of its objects from the database directory
DROP INDEX	Deletes an index
DROP SYNONYM	Deletes a synonym
DROP TABLE	Deletes a table
DROP VIEW	Deletes a view
RUNSTATS	Updates catalog table statistics to optimize SQL access of tables

Deletion of Objects

DROP DATABASE	Deletes a SQL database and all of its objects from the database directory
DROP INDEX	Deletes an existing index
DROP SYNONYM	Deletes a synonym
DROP TABLE	Deletes a table
DROP VIEW	Deletes a view

Embedded SQL

CLOSE	Releases a SQL cursor
DECLARE CURSOR	Creates a cursor and a SELECT command that defines the rows available to the cursor
FETCH	Advances the cursor pointer to the next SELECT result row and copies the row values into dBASE memory variables
OPEN	Opens a cursor and executes the SELECT command associated with the cursor

Query and Update of Data

DELETE	Removes specified rows from a table or updatable view
INSERT	Adds new rows to a table or updatable view
SELECT	Retrieves data from rows of one or more tables or views
UPDATE	Changes the data in selected rows of a table or updatable view

Utility

DBCHECK	Verifies SQL catalog table entries for database tables
DBDEFINE	Creates SQL catalog table entries for database files defined as SQL tables
LOAD DATA	Imports data from a file into a SQL table
RUNSTATS	Updates catalog table statistics to optimize SQL access of tables
UNLOAD DATA	Exports data from a SQL table to another file format

ALTER TABLE

This command adds one or more columns to a table in the current database. If dBASE IV is password-protected, you must be the creator of the table you wish to alter, or you must have the ALTER privilege for the table.

Syntax

ALTER TABLE <table name>
    ADD (<column name><data type>
    [,<column name><data type>...]);

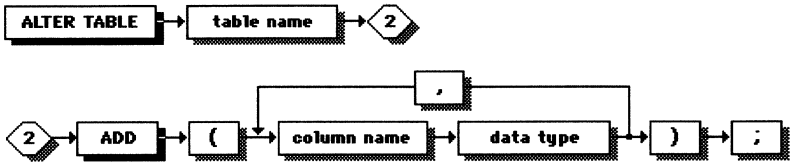


Figure 6-2 ALTER TABLE command

Usage

When you add a column to a table, you must specify its name and data type. For character-type and some numeric-type columns, you also must specify column length and number of decimal places.

Table 6-2 lists the data types that you can use to define columns using the ALTER TABLE command. These are the same data types that you use to define columns in the CREATE TABLE command, and are used according to the same rules. Refer to CREATE TABLE for a more detailed description of SQL data types.

Table 6-2 SQL data types

Data Type	Description
CHAR(n)	Character string data
DATE	Dates (default format, *mm\dd\yy*)
DECIMAL(x,y)	Fixed decimal point numbers
FLOAT(x,y)	Floating point numbers
INTEGER	11-digit integer values
LOGICAL	Logical values (true or false)
NUMERIC(x,y)	Fixed decimal point numbers
SMALLINT	6-digit integer values

NOTE

1. *When adding new columns, remember that the total number of columns in a table cannot exceed 255 and that the sum of the widths of all fields cannot exceed 4,000 bytes.*
2. *dBASE IV SQL does not allow you to create or use dBASE memo fields.*
3. *New character columns added to existing rows are initialized with blank characters. Numeric columns are set to zero. Logical columns are set to false (.F.).*

Each new column is added to the right of the last column that was defined for the table. You can only add columns to a table; you cannot add columns to a view. You cannot remove an existing column or change its definition.

TIP

To remove a column from a table or modify its data type, create a new table with new column definitions and then insert data from the desired columns of the old table using the <SELECT command> form of the INSERT command.

ALTER TABLE

Example

To add two new columns, Phone_no and Last_order, to the Customer table, type:

```
SQL> ALTER TABLE Customer
     ADD (Phone_no CHAR(13), Last_order DATE);
```

See Also

CREATE TABLE, DROP TABLE, INSERT

This command deactivates an open cursor in embedded SQL mode.

Syntax

CLOSE <cursor name>;

Figure 6-3 CLOSE command

Usage

In an embedded SQL program, the DECLARE CURSOR command is used to define the cursor and the SELECT command that creates the cursor's result table. The OPEN command is then used to activate the cursor and execute the SELECT command to produce the result table. The CLOSE command deactivates the named cursor and releases the memory used to hold the result table.

The cursor specified by <cursor name> must already be open. You can reopen a closed cursor by executing another OPEN command. When you reopen a cursor, its SELECT command is re-executed to obtain a new result table.

A cursor is closed automatically:

- When the program returns control to the dot prompt or the SQL prompt.
- When the SET SQL OFF command is executed.
- When the active database is closed using CREATE DATABASE, START DATABASE, or STOP DATABASE.

However, a cursor remains open when an embedded SQL program module calls another module. When execution returns to the calling module, the cursor points to the same row as before the call.

A CLOSE command must be placed after any command that references the cursor, or an error occurs during compilation.

TIP
You can close and reopen a cursor as often as you like to re-execute the cursor's SELECT command. A cursor must be closed before it can be reopened.

Example

The following embedded SQL program DECLAREs the cursor Inv and defines its SELECT command. The OPEN command opens Inv and executes SELECT.

After information has been FETCHed from the SELECT result table, the Invoices program has used the information to generate invoices, and rows for invoiced orders have been deleted from the Sales table, the CLOSE command closes Inv and releases the memory used for the SELECT result table.

```
DECLARE Inv CURSOR FOR
 SELECT Order_no, Sale_date, Staff_no, Cust_no, Invoiced
 FROM Sales
 WHERE NOT Invoiced;
    .
    .
    .
OPEN Inv;
    .
    .
    .
 DO WHILE .T.
    FETCH Inv INTO morder_no, msale_date, mstaff_no, mcust_no, minvoiced;
    IF SQLCODE = 0
      * Print an invoice for order number in the the current row
      * If successful, change minvoiced memory variable to .T.
      *
      DO Invoices WITH morder_no, msale_date, mstaff_no, mcust_no, minvoiced
      *
      IF minvoiced .AND. Sale_date < CTOD("09/22/87")
        DELETE FROM sales
        WHERE CURRENT OF Inv;
      ENDIF
    ELSE
      EXIT
    ENDIF
 ENDDO
CLOSE Inv;
```

See Also

DECLARE CURSOR, FETCH, OPEN

CREATE DATABASE

This command creates a new database to hold new database objects — tables, views, synonyms, and indexes. No authorization is required to create a database.

Syntax

CREATE DATABASE [<path>]<database name>;

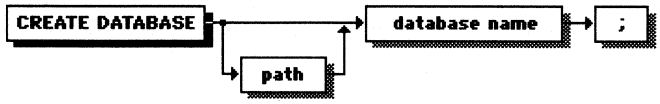


Figure 6-4 CREATE DATABASE command

Usage

The CREATE DATABASE command creates a SQL database and defines a set of SQL catalog tables in the directory that contains the new database. All information about objects created in the database is recorded in the catalog tables.

NOTE
Before using the CREATE DATABASE command, make sure that the SQLHOME =<path name> command is included in your Config.db file. This command designates the directory containing your SQL files. Refer to Getting Started.

The <database name> specification can be any valid dBASE name. That is, the name:

- Can be up to eight characters long.
- Can be composed of letters, numbers, and underscores, but must begin with a letter.
- Cannot duplicate that of another database.

You can specify an explicit operating system path of up to 64 characters. (Do not use spaces in the path specification.) The name and path of the database directory are stored as a row entry in the master catalog table, Sysdbs.

If the database directory specified using <path> does not already exist, a new directory is created in the directory specified by the path. If the database directory already exists, SQL defines it as a new database and copies a set of catalog tables to the directory. If you do not specify a path, the database directory is created as a subdirectory of the current directory.

CREATE DATABASE

WARNING
If you use CREATE DATABASE to define a database with the same name and path specification as your SQLHOME directory (for example, C:\DBASE\SQLHOME), your SQL files may be corrupted.

After you create a SQL database, it is automatically activated. You can also activate a database using the START DATABASE command. To start a database automatically when you enter SQL mode or execute an embedded SQL program, use the SQLDATABASE command in Config.db to specify the database.

If dBASE IV is password-protected, only the creator of a database or the SQLDBA user ID may DROP the database.

Examples

To create the database Mydata as a subdirectory of your current directory, type:

```
SQL. CREATE DATABASE Mydata;
```

To create the database Myapp as a subdirectory of an existing directory named C:\DBASE, type:

```
SQL. CREATE DATABASE C:\Dbase\Myapp;
```

See Also

START DATABASE

CREATE INDEX

This command builds an index based on the values of one or more columns of a table in the current database. If dBASE IV is password-protected, you must be the table's creator or have the INDEX privilege for the table.

Syntax

CREATE [UNIQUE] INDEX <index name>
ON <table name>
(<column name> [ASC/DESC]
[,<column name> [ASC/DESC]...]);

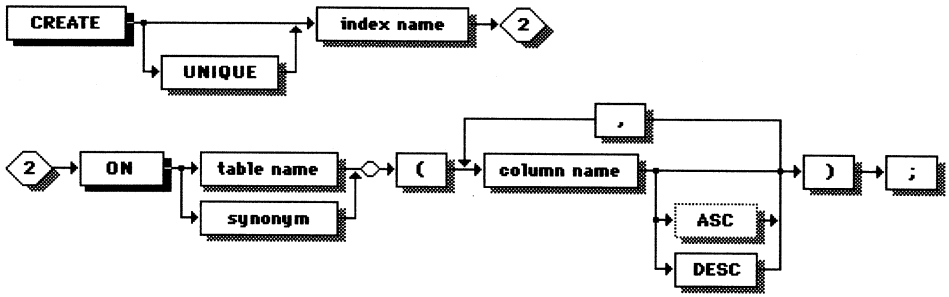


Figure 6-5 CREATE INDEX command

Usage

An index helps you retrieve information quickly from a SQL table. The CREATE INDEX command specifies columns whose values you use in GROUP BY and ORDER BY clauses of data manipulation commands to order result data.

The name of an index defined using the CREATE INDEX command can be any valid dBASE name. That is, the name:

- Can be up to 10 characters long.
- Can be composed of letters, numbers, and underscores, but must begin with a letter.
- Cannot duplicate that of another index in the same database. Each index name in a database must be unique.

In CREATE INDEX, you use the <column name> parameter to specify the name of each index column. You can define an index on a column of any data type except logical. You can list up to 10 columns for an index as long as their combined width does not exceed 100 bytes. If you attempt to include more than 10 <column name> specifications, an error message is displayed.

The ASC and DESC keywords specify ascending or descending order for index rows, based on the values of the indexed column. Because ascending order is the default, you needn't specify ASC when defining an index column. You cannot specify both ascending and descending order in the same index.

If you specify the UNIQUE keyword, dBASE IV verifies that the data in the index columns is unique before creating the index. After a unique index is defined, the system checks INSERT and UPDATE operations and rejects non-unique values supplied for index columns.

You can use CREATE INDEX to create up to 47 different indexes for any table. Each index is defined as a tag within a dBASE .mdx index file that has the same name as the table.

The rows in an index tag contain the values of the indexed column and their locations in the table, arranged in the order that you have specified using CREATE INDEX. Whenever you INSERT, UPDATE, or DELETE table rows, any affected values of the index tag must also be updated.

Although indexes generally improve the performance of SELECTs, they slow INSERT and UPDATE operations when:

- A number of index tags must be updated to reflect changes to the table.
- Changes must be checked for uniqueness.

In SQL, you cannot prescribe the use of a specific index for an operation, as you can in dBASE IV. SQL automatically chooses the most efficient index available.

If dBASE IV is password-protected, only the creator of an index or the SQLDBA user ID can drop the index. However, indexes are dropped automatically when the tables on which they are based are dropped.

NOTE

1. *Indexes can be defined only for tables, not for views. However, when you use a view, dBASE IV SQL uses indexes defined for the tables underlying the view.*
2. *Index rows for a character-type column are arranged by the column's ASCII values, which distinguish between uppercase and lowercase letters.*
3. *You can view information for any index by querying the Sysidxs catalog table (for example, SELECT * FROM Sysidxs WHERE Ixname = "LASTNAME";). You can view the indexes available for tables by displaying columns from the Sysidxs catalog table.*

WARNING
Database files for SQL tables that have unique indexes are available only as read-only files in dBASE mode. To edit or append data in those files, first export the data to another dBASE file using the UNLOAD command or the SAVE TO TEMP clause of the SELECT command.

Examples

To create an ascending order index on the Lastname column of the Customer table, type:

```
SQL. CREATE INDEX Lastname
     ON Customer (Lastname);
```

To create a descending order index on the Unitcost column in the Inventry table, type:

```
SQL. CREATE INDEX Expense
     ON Inventry (Unitcost DESC);
```

To create a unique index for the Order_no column of the Sales table, type:

```
SQL. CREATE UNIQUE INDEX Order_no
     ON Sales (Order_no);
```

See Also

DROP INDEX

CREATE SYNONYM

This command defines a synonym, or alternate name, for a table or view in the current database. No privilege is required to create a synonym. However, if dBASE IV is password-protected, you can use a synonym only to perform operations on a table or view for which you have privileges.

Syntax

CREATE SYNONYM <synonym name>
    FOR <table/view name>;

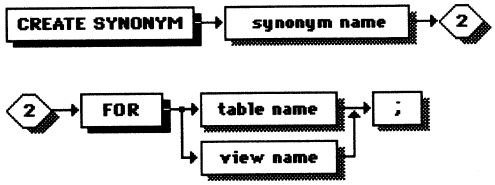


Figure 6-6 CREATE SYNONYM command

Usage

The alternate name defined by the CREATE SYNONYM command can be substituted for the actual name of a table or view in any SQL command except the following: CREATE TABLE, DROP TABLE, CREATE VIEW, DROP VIEW. For example, you could use this command to create an abbreviated name for a table or view with an inconveniently long name.

The name of a synonym defined using the CREATE SYNONYM command can be any valid dBASE name. That is, the name:

- Can be up to eight characters long.
- Can be composed of letters, numbers, and underscores, but must begin with a letter.
- Cannot duplicate that of another synonym, table, or view in the database.

If dBASE IV is password-protected, only the creator of a synonym or the SQLDBA user ID can drop it. Synonyms are automatically dropped when the tables or views on which they are based are dropped.

NOTE
*You can view information for any synonym by querying the Syssyns catalog table (for example, SELECT * FROM Syssyns WHERE Syname = "C1";).*

Examples

To create the synonym C1 for the Customer table, type:

```
SQL. CREATE SYNONYM C1 FOR Customer;
```

Then, to use the synonym, type:

```
SQL. SELECT * FROM C1;
```

See Also

DROP SYNONYM

CREATE TABLE

This command defines a new SQL table in the current database. No authorization is required to create a SQL table. If dBASE IV is password-protected, a new table is encrypted after creation.

Syntax

CREATE TABLE <table name>
    (<column name><data type>
    [,<column name><data type>...]);

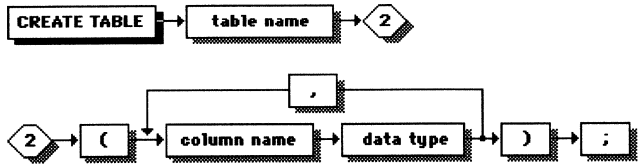


Figure 6-7 CREATE TABLE command

Usage

You use the CREATE TABLE command to specify the name of the new table and the names, data types, and sizes of columns in the new table. The <table name> parameter specifies the name of the table you are creating and can be any valid dBASE name. That is, the name:

- Can be up to eight characters long.
- Can be composed of letters, numbers, and underscores, but must begin with a letter.
- Cannot consist of a single letter from A through J.
- Cannot duplicate that of another table, view, or synonym.

When you create a table, SQL creates a dBASE database (.dbf) file with the same name as the table. If a database file with the same name already exists in the current database directory, a message appears warning that the new file will overwrite the existing file. Fields in the database file correspond to the columns defined for the table.

If SET CATALOG is ON, the new table is added to the open catalog file.

The data types that you use to define table columns are listed in Table 6-3.

Table 6-3 SQL data types

Data Type	Description
SMALLINT	Holds an integer of up to six digits (including sign). Values entered may range from −99,999 to 999,999 (if a plus sign is not specified). This column type maps to a NUMERIC(6,0) field in the database file.
INTEGER	Holds an integer of up to 11 digits (including sign). Values entered may range from −9,999,999,999 to 99,999,999,999 (if a plus sign is not specified). This column type maps to a NUMERIC(11,0) field in the database file.
DECIMAL(x,y)	Holds a signed, fixed decimal point number with *x* total digits (including sign) and *y* decimal places (significant digits to the right of the decimal point). *x* may range from 1 to 19 and *y* may range from 0 to 18. This column type maps to a NUMERIC(x+1,y) field in the database file.
NUMERIC(x,y)	Holds a signed, fixed decimal point number with *x* total digits (including sign and decimal point) and *y* decimal places (significant digits to the right of the decimal point). *x* may range from 1 to 20 and *y* may range from 0 to 18. This column type maps to a NUMERIC(x,y) data type in the database file.
FLOAT(x,y)	Holds a signed, floating point number with *x* total digits (including sign and decimal point) and *y* decimal places (significant digits to the right of the decimal point). *x* may range from 1 to 20 and *y* may range from 0 to 18. The range of numbers you may store is $0.1\ 10^{-307}$ to $0.9\ 10^{+308}$. A number may be specified using scientific (exponential) notation, for example, −9.99E+235. This column type maps to a FLOAT(x,y) field in the database file.
CHAR(n)	Holds a character string of up to *n* characters. *n* may range from 1 to 254. You can enter values to a character-type column from another character-type column or from character-type memory variables or character strings. This column type maps to a CHARACTER(n) field in the database file.

(*continued*)

Table 6-3 SQL data types (*continued*)

Data Type	Description
DATE	Holds a date in the format specified by the SET DATE and SET CENTURY commands. The default format is *mm/dd/yy*. You can enter values to a date-type column from another date-type column, from date-type memory variables, from date strings converted using the dBASE CTOD() function (for example, CTOD("02/15/86")), or from a date delimited by {} (for example, {02/15/86}). This column type maps to a DATE field in the database file.
LOGICAL	Holds a logical true or false value. (.T. represents a true value and .F. represents a false value.) You can enter values to a logical-type column from another logical-type column, from logical-type memory variables, or from the constants .T., .t., .Y., .y., .F., .f., .N., and .n.. This column type maps to a LOGICAL field in the database file.

NOTE

1. *The total number of columns in a table cannot exceed 255, and the total width of a table cannot exceed 4,000 bytes.*
2. *dBASE IV SQL does not allow you to create or use dBASE memo fields.*
3. *A column name can be up to 10 characters long and can consist of letters, numbers, and underscore characters. A name cannot contain a blank and must begin with a letter.*

Examples

To define a table, Shipment, to record shipments leaving a firm, type:

```
SQL. CREATE TABLE Shipment
     (Ship_no      CHAR(6),
      Shipdate     DATE,
      Order_no     CHAR(6),
      Shipper      CHAR(25),
      Weight       DECIMAL(4,1),
      Value        DECIMAL(10,2));
```

See Also

ALTER TABLE, DBDEFINE, DROP TABLE, INSERT, LOAD DATA

CREATE VIEW

This command creates a *virtual* table from columns defined in one or more base tables (including catalog tables) or views. If dBASE IV is password-protected, you must be the creator of, or have the SELECT privilege on, every table referenced in the view's definition.

Syntax

CREATE VIEW <view name> [(<column name>, <column name>...)]
    AS <SELECT command>
    [WITH CHECK OPTION];

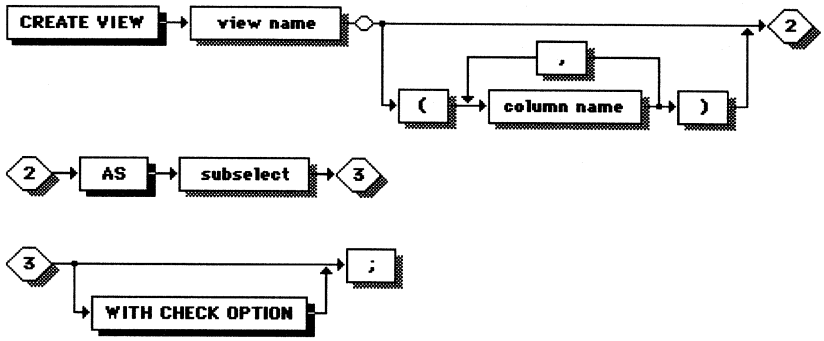


Figure 6-8 CREATE VIEW command

Usage

The name of a view defined using the CREATE VIEW command can be any valid dBASE name. That is, the name:

- Can be up to eight characters long.
- Can be composed of letters, numbers, and underscores, but must begin with a letter.
- Cannot consist of a single letter from A through J.
- Cannot duplicate that of another view, table, or synonym. (A view name can be the same as that of an existing index, but this is not recommended.)

A view definition is stored in the SQL catalog table Sysviews.

A view is called a virtual table because it does not actually contain stored data. Instead, the view contains the *definition* of the data, as derived from one or more base tables or views. If a view is based on another view, its data is derived from the base tables of the other view.

You can use a SELECT query to display view information just as you do table information. As information in the view's base tables changes, so does the information displayed using the view.

Once a view is created, only the creator of the view or the SQLDBA user ID can delete it. However, if any of the view's base tables or views is dropped, the view itself is automatically dropped.

Defining View Data

The CREATE VIEW command employs a SELECT command to extract column values from the base tables or views. Data is transferred from the base table or view columns specified in the AS <SELECT command> clause to corresponding view columns.

You can specify any valid SELECT command in creating a view, with the following restrictions:

1. You cannot specify a UNION.
2. You cannot include FOR UPDATE OF, INTO, ORDER BY, or SAVE TO TEMP clauses.
3. If you specify a GROUP BY clause, you cannot join the view with another table or view.

Naming Columns in a View

The optional <column name> specification defines column names for a view. Otherwise, view column names are the same as those for the base tables. You must specify view column names when:

- Base table columns are derived using constants, character strings, memory variables, expressions, or SQL or dBASE functions.
- Column names in two or more base tables are identical.

NOTE
Naming conventions for view columns are the same as those for table columns (refer to the third note under CREATE TABLE). The size limit (number of rows and columns) for a view is the same as for tables (refer to the first note under CREATE TABLE).

Querying a View

You can use any valid SELECT command to query a view. However, once a SELECT command that is valid for querying a table is expanded for the view definition, the command may be invalid for the view.

For example, consider the following CREATE VIEW command:

```
CREATE VIEW Staff1 (C1, C2)
  AS SELECT AVG(Salary), Location
  FROM Staff
  GROUP BY Location;
```

The query

```
SELECT C1 FROM Staff1;
```

is invalid for view Staff1 because in its expansion,

```
SELECT AVG(Salary) FROM Staff
GROUP BY Location;
```

the GROUP BY column, Location, is not included in the SELECT clause. However, the following queries are valid for view Staff1:

```
SELECT * FROM Staff1;
```

```
SELECT C2 FROM Staff1;
```

Using an Updatable View

If a view is *updatable*, you can insert, update, or delete information in the base table by inserting, updating, or deleting data in the view. A view is updatable if:

- It has a single base table
- It does not define a self-join of the base table
- It is based on an updatable view

To create an updatable view, you must have the INSERT, UPDATE, or DELETE privilege for the base table or updatable view.

A view is not updatable if the SELECT command used to define it contains:

- The DISTINCT keyword, a SQL aggregate function, or a dBASE function in the SELECT clause
- A FROM clause naming a non-updatable view or more than one table or view

- A GROUP BY clause
- A nested subquery containing a FROM clause that references the view's base table

If a view column is derived using an arithmetic expression or constant, you cannot update that column.

You can specify WITH CHECK OPTION in the CREATE VIEW command for an updatable view. This causes rows that are inserted, updated, or deleted using the view to be checked against the WHERE condition of the view's SELECT command. If any row does not satisfy this condition, the row is not allowed to affect the base table.

Examples

To create a view of all columns from the Sales table that includes only rows for non-invoiced orders, type:

```
SQL. CREATE VIEW Billings
     AS SELECT *
     FROM Sales
     WHERE NOT Invoiced;
```

To create an updatable view that contains a subset of columns from the Staff table, type:

```
SQL. CREATE VIEW La
     AS SELECT Staff_no, Lastname, Firstname, Hiredate
     FROM Staff
     WHERE Location = "LOS ANGELES";
```

To create a view that combines the Lastname and Firstname columns of the Customer table, type:

```
SQL. CREATE VIEW Address
     (Fullname, City, State)
     AS SELECT Firstname+Lastname, City, State
     FROM Customer;
```

To create a view that combines column data from two different tables, type:

```
SQL. CREATE VIEW Orders
     (Order_no, Sale_date, Seller)
     AS SELECT Sales.Order_no, Sale_date, Lastname
     FROM Sales, Staff
     WHERE Sales.Staff_no = Staff.Staff_no;
```

Using the La view defined above, revise the spelling of Zambini's first name in the Staff table:

```
.SQL UPDATE La
        SET Firstname = "Richard"
        WHERE Lastname = "Zambini";
```

See Also

DROP VIEW, SELECT

DBCHECK

This command verifies that the catalog table entries for one or all tables in the current database are consistent with the tables' underlying .dbf and .mdx file structures.

Syntax

DBCHECK [<table name>];

Figure 6-9 DBCHECK command

Usage

If you enter the DBCHECK command without the <table name> parameter, DBCHECK verifies the files for all SQL tables and their associated indexes in the current database. If you specify a table name, DBCHECK checks the .dbf and .mdx files only for that table.

DBCHECK returns an error message for every table for which the .dbf or .mdx file structures do not match the definitions in the SQL database's catalog tables. If you receive error messages, perform the following steps:

1. Copy the .dbf and .mdx files named in the error messages from the current database directory to another directory or backup disk.
2. In SQL mode, DROP the tables for which errors were displayed to remove catalog table entries for those tables.
3. Copy the .dbf and .mdx files back into the database directory.
4. In SQL mode, enter the DBDEFINE command for each .dbf file you want to redefine as a SQL table. Associated indexes are automatically redefined as part of the operation of DBDEFINE.

If the database file for any table in the current database is already open in dBASE mode when DBCHECK is executed, you will receive a file-opener error.

NOTE
DBCHECK cannot check files that were encrypted in dBASE mode using the PROTECT command.

Examples

To verify catalog table entries for the Staff table, type:

```
SQL. DBCHECK Staff;
```

To verify entries for all tables in the Samples database, type:

```
SQL. DBCHECK;
```

See Also

DBDEFINE

DBDEFINE

This command creates catalog table entries for one or more dBASE database files and associated multiple index files in the current database. No privilege is needed to execute DBDEFINE. If dBASE IV is password-protected, during DBDEFINE new tables are encrypted by SQL and the current user ID is named as creator of the tables.

Syntax

DBDEFINE [<.dbf file name>];

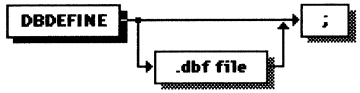


Figure 6-10 DBDEFINE command

Usage

When you enter the DBDEFINE command without the <.dbf filename> parameter, DBDEFINE creates catalog entries for all .dbf and associated .mdx files in the current SQL database that are not already defined as tables and indexes. When you enter DBDEFINE with a filename, DBDEFINE creates catalog entries only for the named .dbf file and associated .mdx files.

NOTE
To make sure that catalog table entries for indexes will reflect the current state of their database files after you have executed DBDEFINE, REINDEX the files to be DBDEFINEd before entering SQL mode, or execute the SQL RUNSTATS command after executing DBDEFINE.

You cannot use DBDEFINE to create catalog table entries for:

- An .ndx index file. You must first convert the file to an .mdx tag.
- An .mdx tag whose index key expression would result in creation of a SQL index on more than 10 columns.
- A UNIQUE index created in dBASE mode.
- Memo fields in a database file. (You cannot access memo fields while in SQL mode.)
- dBASE views. You must use the CREATE VIEW command to re-create a view in SQL mode after you have DBDEFINEd the view's base tables.

- A database file that was encrypted in dBASE mode. To unencrypt a file before entering SQL mode and executing DBDEFINE, SET ENCRYPTION OFF and use the COPY TO command to create an unencrypted version of the file. To do this, you must have sufficient PROTECT privileges to access the original file.
- A SQL-encrypted database file. Use the UNLOAD DATA command to create an unencrypted version of the file before executing DBDEFINE.
- A database file that is open in dBASE mode (you will receive a file open error when executing DBDEFINE). Use the CLOSE command to close the file before executing DBDEFINE.
- A file whose name is a single letter from A through J.

DBDEFINE displays error messages identifying files for which it cannot define tables or indexes, files that it cannot read, and files for which catalog entries already exist.

Although you cannot access memo fields in SQL, associated memo (.dbt) files must be available in the database directory when their database file is DBDEFINEd. (If memo files are not available, you will receive a file open error.) Before DBDEFINEing such a database file, therefore, you must do one of the following:

- Copy memo files into the database directory.
- Enter the USE command for the database file to create empty memo files.
- Use the dBASE MODIFY STRUCTURE command to display the database design screen and remove the memo field.

Examples

To create catalog entries for the Detail file, type:

```
SQL. DBDEFINE Detail;
```

To create catalog entries for all database files that are not defined as tables in the current database directory, enter DBDEFINE without a file name, for example:

```
SQL. DBDEFINE;
```

See also

CREATE DATABASE, DBCHECK, RUNSTATS, START DATABASE

DECLARE CURSOR

This command defines a cursor and an associated SELECT command. A cursor enables a program in embedded SQL mode to execute the SELECT command and then process rows from its result table, one at a time. If dBASE IV is password-protected, you must have privileges on the objects named in the SELECT command.

DECLARE <cursor name> CURSOR
    FOR <SELECT command>
    [FOR UPDATE OF <column list>/<ORDER BY clause>];

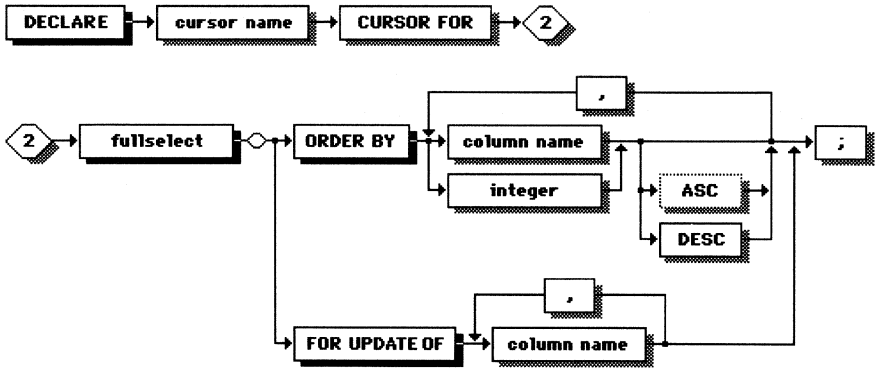


Figure 6-11 DECLARE CURSOR command

Usage

An embedded SQL program uses a cursor to point to rows in a result table, one at a time. The data in the row to which the cursor is pointing is transferred to dBASE memory variables for processing.

The program first executes the cursor's SELECT command using an OPEN <cursor> command to produce a result table. A FETCH command advances the cursor to each row of the result table, in turn, and transfers row data to the memory variables defined by the FETCH.

Once declared and opened, a cursor remains defined and open until:

- The program executes the CLOSE <cursor> command, leaves SQL mode, or finishes executing. (Thus, a cursor remains open during execution of lower-level subroutines, whether these be dBASE program (.prg) files or SQL program (.prs) files.)
- The database in which it is opened is stopped using a STOP DATABASE or DROP DATABASE command, a START DATABASE command for another database, or a CREATE DATABASE command that activates a new database.

DECLARE CURSOR

A cursor enables a program to update or delete rows in the base table that correspond to rows in the SELECT result table:

- The FOR UPDATE OF clause of the cursor's SELECT command enables a subsequent UPDATE...WHERE CURRENT OF command to update values in the row of the base table that corresponds to the result row to which the cursor is pointing.
- The WHERE CURRENT OF clause of the DELETE command enables the program to delete the base table row that corresponds to the result row to which the cursor is pointing.

NOTE

1. *A SELECT command that includes FOR UPDATE OF cannot also include an ORDER BY or UNION clause.*
2. *Rows cannot be updated using UPDATE...WHERE CURRENT OF if the cursor's SELECT command contains aggregate functions or the DISTINCT keyword, or GROUP BY, HAVING, or UNION clauses. The FROM clause cannot specify more than one table, a view that is not updatable, or a table that is also specified in a subquery within the SELECT command.*
3. *A cursor's SELECT command cannot contain an INTO or SAVE TO TEMP clause.*

A cursor created in one SQL program file cannot be referenced by another SQL program file. If a cursor created by one procedure is referenced by other procedures in the same program file, the creating procedure must physically precede the other procedures.

You cannot execute a DECLARE CURSOR command conditionally using the IF...ENDIF program construct.

Examples

To declare a cursor to display rows from the Sales table for uninvoiced orders, you could enter the following command in a program:

```
DECLARE Inv CURSOR FOR
 SELECT Order_no, Sale_date, Staff_no, Cust_no, Invoiced
 FROM Sales
 WHERE NOT Invoiced;
```

The following example expands this cursor to allow updating of rows in the Sales table:

```
DECLARE Inv CURSOR FOR
   SELECT Order_no, Sale_date, Staff_no, Cust_no, Invoiced
   FROM Sales
   WHERE NOT Invoiced
   FOR UPDATE OF Invoiced;
      .
      .
      .
OPEN Inv;
 DO WHILE .T.
   FETCH Inv INTO morder_no, msale_date, mstaff_no, mcust_no, minvoiced;
   IF SQLCODE = 0
            * Print an invoice for order number in the current row
            * If successful, change minvoiced memory variable to .T.
            *
      DO Invoices WITH morder_no, msale_date, mstaff_no, mcust_no, minvoiced
            *
       IF minvoiced
          UPDATE Sales
          SET Invoiced = .T.
          WHERE CURRENT OF Inv;
            *
       ENDIF
   ELSE
      EXIT
   ENDIF
 ENDDO
CLOSE Inv;
```

See Also

CLOSE, FETCH, OPEN

DELETE

This command deletes specified rows from a table. If dBASE IV is password-protected, you must have the DELETE privilege on the table or the updatable view that is used to access the table.

There are two forms of the DELETE command. The first form, normally used with a WHERE clause, can be executed in interactive or embedded SQL mode. The second form, which includes the WHERE CURRENT OF clause, is used exclusively in embedded SQL mode with a declared cursor.

Syntax

DELETE (1)
FROM <table name> [<alias name>]
[<WHERE clause>];

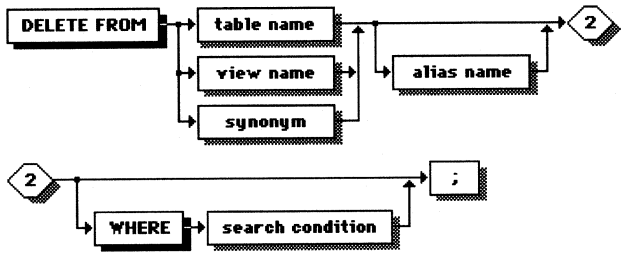


Figure 6-12 DELETE...WHERE <clause> form of command

DELETE (2)
FROM <table name>
WHERE CURRENT OF <cursor name>;

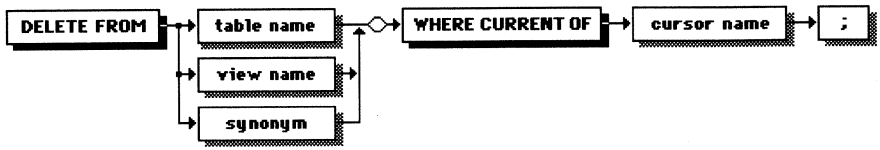


Figure 6-13 DELETE...WHERE CURRENT OF form of command

Usage

If you use a view to delete rows in a base table, the view must be updatable (refer to the CREATE VIEW command).

DELETE with WHERE Clause

The first form of the DELETE command uses a WHERE clause to select the rows to be deleted. The WHERE clause can contain simple search conditions, combined search conditions, and simple (non-correlated) subqueries. A WHERE subquery cannot reference the same table or view from which you are deleting rows.

TIP
To verify that a WHERE condition will cause the correct rows to be deleted, execute the SELECT command with the same WHERE clause.

WARNING
Unlike the dBASE DELETE command, which only marks records for deletion, the SQL DELETE command immediately removes the rows from the table. If you execute DELETE without a WHERE clause and SET SAFETY is ON, a warning message is displayed. If you choose to execute the command, all rows in the table are deleted.

In a program, if you execute the DELETE command within a transaction (using the BEGIN/END TRANSACTION command), deleted rows are not immediately removed from the table. Until the transaction ends, these rows are displayed in result tables, preceded by an asterisk.

To exempt deleted rows from being processed or displayed while the transaction is still in progress, use the dBASE SET DELETED ON commmand. This is particularly important if others are simultaneously viewing the same data on a network.

DELETE with WHERE CURRENT OF Clause

The second form of the DELETE command is used only in an embedded SQL program. DELETE...WHERE CURRENT OF deletes rows in a base table that correspond to result rows pointed to by a cursor.

In the program, the following commands must occur in the order specified:

1. DECLARE CURSOR — Defines the cursor subsequently referenced in the WHERE CURRENT OF clause; defines a SELECT command that references the table or updatable view from which rows are to be DELETEd.
2. OPEN — Executes the cursor's SELECT command to produce the result rows that correspond to the rows that are to be DELETEd.

3. FETCH — Positions the cursor to each result row, in turn; moves row values into dBASE memory variables.
4. DELETE...WHERE CURRENT OF — Deletes the row in the base table that corresponds to the result row pointed to by the cursor, depending on the FETCHed values of memory variables.

The IF...ELSE program construct is used to evaluate memory variables and determine whether the current row is to be deleted. The IF condition establishes the condition for deletion (for example, IF the minvoiced memvar is set to .T.).

When using DELETE in a program, include the command in a transaction (using the dBASE BEGIN/END TRANSACTION command). This allows you to use the dBASE ROLLBACK command to restore the table to its previous state if the transaction does not complete successfully, and to preserve the integrity of the database for other users on a network.

NOTE
Rows deleted during a transaction using DELETE...WHERE CURRENT OF are not removed from the table until the end of the transaction in which the associated cursor is closed. If the cursor is OPENed and CLOSEd outside of the transaction, rows are not removed until the cursor is closed.

Examples

To delete a row from the Staff table, type:

```
SQL. DELETE FROM Staff
     WHERE Lastname = "Long";
```

The system displays the message **1 row(s) deleted**.

To delete any row in the Sales table whose Sale_date column contains a date earlier than 9/22/87, type:

```
SQL. DELETE
     FROM Sales
     WHERE Sale_date < {09/22/87};
```

The system displays the message **4 row(s) deleted**.

The following example illustrates the use of DELETE..WHERE CURRENT OF with a cursor:

```
DECLARE Inv CURSOR FOR
 SELECT Order_no, Sale_date, Staff_no, Cust_no, Invoiced
 FROM Sales
 WHERE NOT Invoiced;
    .
    .
    .
OPEN Inv;
 DO WHILE .T.
    BEGIN TRANSACTION
    FETCH Inv INTO morder_no, msale_date, mstaff_no, mcust_no, minvoiced;
    IF SQLCODE = 0
       * Print an invoice for order number in the the current row
       * If successful, change minvoiced memory variable to .T.
       *
       DO Invoices WITH morder_no, msale_date, mstaff_no, mcust_no, minvoiced
       *
       IF minvoiced .AND. Sale_date < {09/22/87}
          DELETE FROM Sales
          WHERE CURRENT OF Inv;
       ENDIF
    ELSE
       EXIT
    ENDIF
    END TRANSACTION
 ENDDO
CLOSE Inv;
```

See Also

DECLARE CURSOR, FETCH, SELECT

DROP DATABASE

This command drops (deletes) a database. If dBASE IV is password-protected, only the creator of a database or the SQLDBA user ID can drop the database.

Syntax

DROP DATABASE <database name>;

Figure 6-14 DROP DATABASE command

Usage

The DROP DATABASE command drops a database by deleting:

- Every .dbf and .mdx file in the database directory for which there is a corresponding entry in the Systables and Sysidxs catalog tables.
- The entry for the database in the Sysdbs catalog table.
- Entries in the open dBASE catalog file for tables in the database (if SET CATALOG is ON).

DROP DATABASE does not delete the directory that contains the database.

Before you can drop a database, you must deactivate it using the STOP DATABASE command. When you execute DROP DATABASE, a prompt box appears warning you that all SQL tables in the database will be dropped. To cancel the operation, select the **Cancel** option from the box.

If the database specified by DROP DATABASE cannot be found, an error message appears. If you choose to proceed, the command drops whatever data is found, including references to the database in the Sysdbs master catalog table.

NOTE
On a network, you cannot DROP a database that another user has activated.

See Also

CREATE DATABASE, SHOW DATABASE, START DATABASE, STOP DATABASE

DROP INDEX

This command drops (deletes) an index that has been defined on the columns of a table. If dBASE IV is password-protected, only the creator of an index or the SQLDBA user ID can drop the index.

Syntax

DROP INDEX <index name>;

Figure 6-15 DROP INDEX command

Usage

The DROP INDEX command is used to remove a specific index. Indexes are automatically dropped when the table on which they are defined is dropped.

See Also

CREATE INDEX, DROP TABLE

DROP SYNONYM

This command drops (deletes) a table or view synonym (alternate name). If dBASE IV is password-protected, only the creator of a synonym or the SQLDBA user ID can drop the synonym.

Syntax

DROP SYNONYM <synonym name>;

Figure 6-16 DROP SYNONYM command

Usage

The DROP SYNONYM command is used to remove a specific synonym. A synonym is automatically dropped when the table or view for which it is defined is dropped.

See Also

CREATE SYNONYM

DROP TABLE

This command drops (deletes) a table. If dBASE IV is password-protected, only the creator of a table or the SQLDBA user ID can drop the table.

Syntax

DROP TABLE <table name>;

Figure 6-17 DROP TABLE command

Usage

Executing the DROP TABLE command deletes:

- The database file that contains the table's data.
- Any reference to the table in the SQL catalog tables.
- All indexes, views, and synonyms that are based on the table.
- The entry for the table from the open dBASE catalog file (if SET CATALOG is ON).

The <table name> parameter cannot be a synonym for the table.

WARNING
You cannot restore a table dropped using the DROP TABLE command. If you drop a table by mistake, you must re-create the table, insert data, and re-create associated objects.

See Also

CREATE TABLE, DELETE

DROP VIEW

This command drops (deletes) a view. If dBASE IV is password-protected, only the creator of the view's base table or the SQLDBA user ID can drop the view.

Syntax

DROP VIEW <view name>;

Figure 6-18 DROP VIEW command

Usage

Executing the DROP VIEW command to delete a view deletes:

- The view's definition in the SQL catalog tables.
- All synonyms and views based on the view.

Dropping a view does not affect the tables or views on which the view is based.

A view and its synonyms are automatically dropped when the view's base tables are dropped.

The <view name> parameter cannot be a synonym for the view.

See Also

CREATE VIEW, DROP TABLE

This command positions a cursor to each row of the cursor's result table in turn, and transfers the values from that row into corresponding dBASE memory variables. This command can be used only in an embedded SQL program, not interactively.

Syntax

FETCH <cursor name>
    INTO <memvar list>;

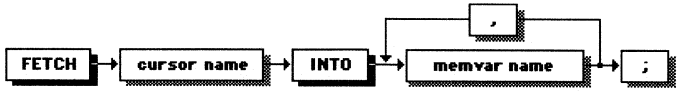


Figure 6-19 FETCH command

Usage

The FETCH command enables a program to process row values from the result table associated with a declared cursor. In the program, FETCH operates with DECLARE CURSOR and OPEN commands in the following sequence:

1. DECLARE CURSOR creates the cursor and specifies an associated SELECT command.
2. OPEN executes the SELECT command to produce a result table.
3. FETCH creates the memory variables specified by <memvar list>, advances the cursor through the result table one row at a time, and moves each row's values into the memory variables.

Each memory variable specified in a FETCH memvar list must correspond, from left to right, to a column in the cursor's SELECT result table. If the cursor's SELECT command does not return any rows, the FETCH memory variables are not created.

If dBASE IV is password-protected, privileges on the table named in the SELECT command's FROM clause are checked for the user executing the program before the DECLARE CURSOR or OPEN command is executed.

When the FETCH command is first executed, it positions the cursor to the first row of the result table. Thereafter, each time the command is executed the cursor is advanced to the next row until the last row of the result table is reached.

NOTE
The cursor can only be advanced in a forward direction. To reset the cursor to the first row of the result table without advancing it to the last result row, CLOSE the cursor and then re-OPEN it.

Your program can use the SQL system status variables, Sqlcnt and Sqlcode, to determine the status of a FETCH operation. When OPEN executes the SELECT command, Sqlcnt contains the number of rows in the result table. If the SELECT returns no rows, Sqlcnt contains 0 and Sqlcode contains a value of 100.

TIP
Because the cursor's SELECT may not return any rows, have your program check the value of Sqlcnt before attempting to process FETCH memory variables.

After FETCH successfully advances the cursor and transfers row values into memory variables, Sqlcode contains 0. If the FETCH is unsuccessful, Sqlcode contains -1. If the last result row has already been processed, Sqlcode contains 100, indicating that there are no more rows to process.

Example

To FETCH Order_no, Sale_date, Staff_no, Cust_no, and Invoiced columns from selected rows of the Sales table, your SQL program could include the following commands:

```
DECLARE Inv CURSOR FOR
 SELECT Order_no, Sale_date, Staff_no, Cust_no, Invoiced
 FROM Sales
 WHERE NOT Invoiced;
    .
    .
    .
OPEN Inv;
    .
    .
    .
 IF SQLCODE > 0
   DO WHILE .T.
      FETCH Inv INTO  morder_no, msale_date, mstaff_no, mcust_no, minvoiced;
      IF SQLCODE > 0
           .
           .
           .
      ELSE
         EXIT
      ENDIF
   ENDDO
 ENDIF
```

The IF procedure checks the value of Sqlcnt to verify that there are result rows to be processed. If there are not, the program ends.

See Also

DECLARE CURSOR, DELETE, FETCH, UPDATE

GRANT

This command assigns access privileges on tables and views in the current database to user IDs created with the dBASE PROTECT command.

Syntax

GRANT ALL [PRIVILEGES]/<privilege list>
    ON [TABLE] <table list>
    TO PUBLIC/<user list>
    [WITH GRANT OPTION];

The privilege list consists of one or more of the following:

    ALTER
    DELETE
    INDEX
    INSERT
    SELECT
    UPDATE [(<column list>)]

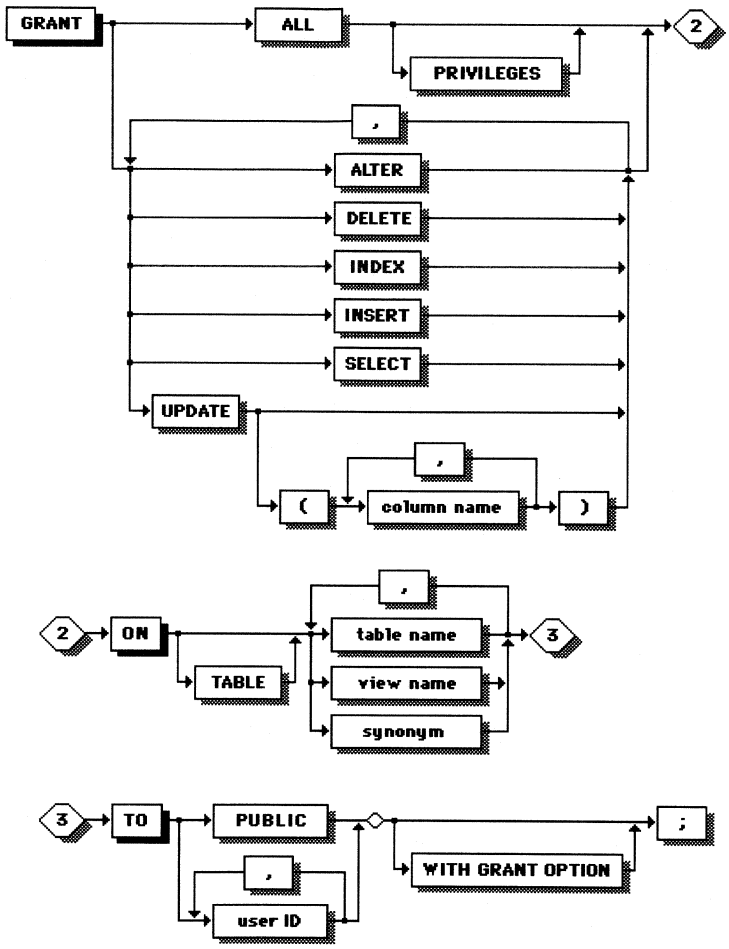


Figure 6-20 GRANT command

Usage

SQL provides two commands, GRANT and REVOKE, to assign or remove user privileges on tables or views in a database. All privileges controlled by GRANT and REVOKE are recorded in the SQL catalog tables for the database.

GRANT

GRANT and REVOKE operate in conjunction with dBASE IV password protection, which is installed by a database administrator using the PROTECT command. Once dBASE IV is password-protected, each user must log in to dBASE IV with a user ID. When a user switches to SQL mode after log-in, the operations that can be performed in each database are dictated by the privileges assigned the user's user ID.

The GRANT command lets you assign privileges on tables and views to other users. The privileges that you can assign are:

- ALTER — Ability to add columns to a table (cannot be granted on a view)
- DELETE — Ability to delete rows from a table or view
- INDEX — Ability to use the CREATE INDEX command (cannot be granted on a view)
- INSERT — Ability to add rows to a table or view
- SELECT — Ability to display rows from a table or view
- UPDATE — Ability to update any row value in a table or view, or only values for specific columns

You can GRANT privileges:

- All at once, using the ALL keyword (which cannot be used for a view)
- Individually, using <privilege list>, which names privileges to be granted, separated by commas
- ON one or more tables or views, using <table list>, which names tables and views on which privileges are to be granted, separated by commas
- TO all users at once, using the PUBLIC keyword
- TO individual users, using the <user list>, which specifies the user IDs to be granted privileges, separated by commas

The PRIVILEGES and TABLE keywords are used optionally for clarity.

NOTE
In a password-protected database, only the creator of the table and the SQLDBA user ID have privileges on the table. Either user ID can GRANT privileges on the table (except for the DROP privilege) or DROP the table. The SQLDBA user ID possesses all privileges on all tables, views, synonyms, and indexes in every database.

GRANTed privileges are cumulative; that is, you can use GRANT to add privileges to those a user already has. The same privilege can be granted to a user more than once.

If you include the WITH GRANT OPTION clause in the GRANT command, grantees are able to grant to others the privileges they've been granted. If you later REVOKE the privileges you GRANTed to these users, these same privileges that the users granted to other users are likewise REVOKEd.

NOTE
WITH GRANT OPTION:

- *Cannot be specified in a command that grants privileges to PUBLIC.*
- *Is ignored if the UPDATE privilege is specified with the <column list> option.*

Examples

To grant all privileges on the Staff table to user ID Janice, type:

```
SQL. GRANT ALL ON Staff
       TO Janice;
```

To grant INSERT, UPDATE, and DELETE privileges on the Staff table to user ID Greg, type:

```
SQL. GRANT INSERT, UPDATE, DELETE ON Staff
       TO Greg;
```

See Also

REVOKE

INSERT

This command inserts rows in a table or updatable view. There are two forms of the command; either can be executed interactively or embedded in a SQL program. If dBASE IV is password-protected, you must have the INSERT privilege on the specified table or view.

Syntax

INSERT INTO <table name> (1)
    [(<column list>)]
    VALUES (<value list>);

INSERT INTO <table name> (2)
    [(<column list>)]
    <SELECT command>;

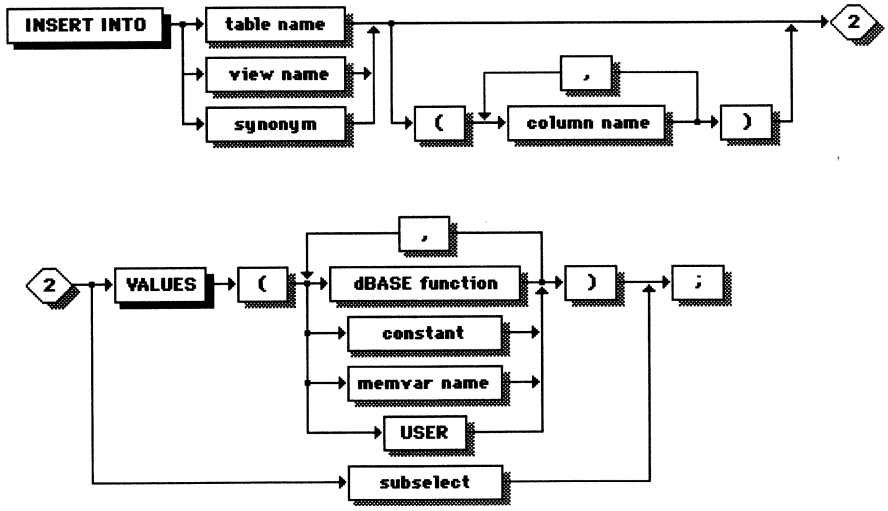


Figure 6-21 INSERT command

Usage

You use the INSERT command to insert rows in a table or updatable view (refer to the CREATE VIEW command). You can specify the table or view name or use a synonym for the name.

In the first form of the command, you use <value list> to specify individual column values that are to be inserted for the new row. In the second form, you insert column values for one or more rows that a SELECT command has retrieved from another table.

The values that are specified in a value list:

- Can be constants, dBASE memory variables, dBASE functions that return a value, or the SQL keyword USER (which returns the user ID of the current user).
- Are separated by commas. Character constants are enclosed by single or double quotes (′ or ″) and dates by curly braces ({}) or the CTOD() function.
- Must be of the same data type as that defined for the column in which you are inserting it. (For example, ''Zambini'' must be inserted in a character-type column, {04/15/88} in a date-type column.)

If you are not inserting a value for each column in the table or view, each unspecified value is initialized as follows:

- Character column, with a blank
- Numeric column, with a zero
- Date column, with a blank date string (format mm/dd/yy)
- Logical column, with the value .F.

You use the <column list> option to specify the names of the table or view columns into which row values are to be inserted. You can omit a column list if the order of the values specified in the value list, or the order of column names specified in the SELECT clause, is the same as the order in which the corresponding columns are defined in the table or view into which you are inserting.

However, you must include a column list if you are not specifying (or SELECTing) a value for every column in the table or view. A column list need not be specified in the same order as table or view columns are defined. However, the column list must have the same order as that of the value list (or of the SELECT clause) that specifies the corresponding column values.

Examples

To insert a single row in the Staff table, type:

```
SQL. INSERT INTO Staff
     VALUES ("000013", "Ross", "Terry", {07/15/87},
     "LOS ANGELES", "000001", 5678, 3.5);
```

The system displays the message **1 row(s) inserted**. Note that this command does not require a column list because values are inserted in the same order as their corresponding columns are defined in the Staff table.

To insert a number of rows from one table into another table that has the same column structure, type:

```
SQL. INSERT INTO LA
     SELECT *
     FROM Staff
     WHERE Location = "LOS ANGELES";
```

See Also

CREATE TABLE, CREATE VIEW, SELECT

This command imports data from an external file and appends it to an existing SQL table in the current database. If dBASE IV is password-protected, you must be the creator of, or have the INSERT privilege for, the table in which data is LOADed.

Syntax

LOAD DATA FROM [path]<file name>
INTO TABLE <table name>
[[TYPE] SDF/DIF/WKS/SYLK/FW2/RPD/DBASEII/
DELIMITED [WITH BLANK/WITH <delimiter>]];

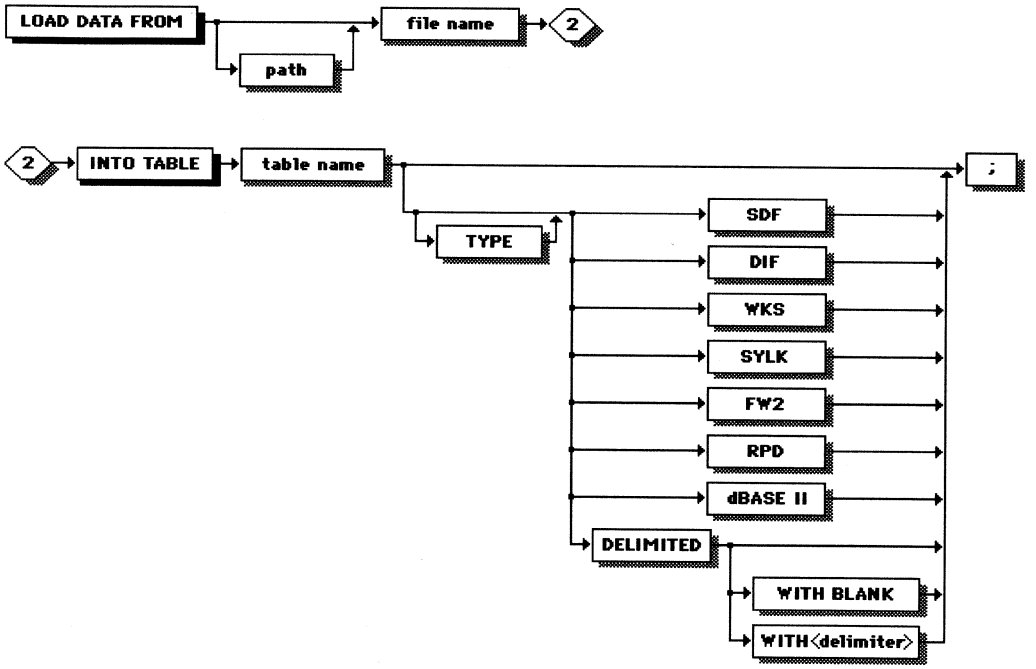


Figure 6-22 LOAD DATA command

LOAD DATA

Usage

The LOAD DATA command supports the same file formats as the dBASE APPEND FROM command. You can use the LOAD DATA command to import data from the following types of files:

- dBASE II database files (dBASEII)
- dBASE III, dBASE III PLUS, and dBASE IV database files
- RapidFile database files (RPD)
- Framework II database and spreadsheet files (FW2)
- Delimited format ASCII files (DELIMITED)
- System data format ASCII files (SDF)
- VisiCalc format files (DIF)
- Multiplan spreadsheet format files (SYLK)
- Lotus 1-2-3 format files (WKS)

The file from which data is loaded can be in the current DOS directory or in another directory specified by an explicit DOS path. If you do not specify a file TYPE, import from a database (.dbf) file is assumed.

The following rules govern the import process:

- The SQL table to which data is appended must already exist.
- The table must have as many columns as the number of fields expected from the import file (unless the import file is a dBASE database file).
- The table's column definitions must match, in length and data type, the imported fields. If length does not match, field data is truncated. If data type does not match, an error is reported.

The following rules govern import from dBASE database files:

- The table's column names must match the file's field names. However, the number of columns need not match the number of fields. Data is imported only for fields whose names correspond to table column names.
- Memo fields are not imported.
- The import file must not be encrypted, either by dBASE IV or SQL.
- If the import file is a dBASE II database file, you must specify the dBASE II file type.

If you're importing data from spreadsheets or character-delimited files, the SQL table structure must match the format of the import file. Spreadsheets should be stored in *row-major* order (as opposed to *column-major* order).

The DELIMITED option specifies import from a delimited format ASCII text (.txt) file. The LOAD DATA command makes the following assumptions about the import file based on your DELIMITED specification:

- DELIMITED — File fields are separated by commas and character fields are delimited by double quotes. This can also be expressed by DELIMITED WITH ".
- DELIMITED WITH BLANK — File fields are separated by one space and character fields are not delimited.
- DELIMITED WITH <delimiter> — File fields are separated by commas and character fields are delimited by the specified delimiter. Double quotes is the default delimiter.

NOTE
If PROTECT is installed:

- *Data is automatically encrypted by SQL when it is loaded using LOAD DATA. (File data created using UNLOAD DATA is not encrypted.)*
- *You cannot use LOAD DATA to import data from a database file that has been encrypted by SQL or dBASE IV; however, you can import data from an unencrypted SQL table.*

Example

To append data from a RapidFile file to the Staff table, type:

```
SQL. LOAD DATA FROM \RAPID\RFDATA\Salstaff.rpd
     INTO TABLE Staff TYPE RPD;
```

See Also

CREATE TABLE, UNLOAD DATA

OPEN

This command opens a previously DECLAREd cursor, executes the associated SELECT command, and positions the cursor above the first row in the result table. This command can only be executed in an embedded SQL program.

Syntax

OPEN <cursor name>;

Figure 6-23 OPEN command

Usage

If dBASE IV is password-protected, the user executing the embedded SQL program must have adequate privileges on the tables and views named in the SELECT command of the DECLARE CURSOR command.

Once opened, a cursor remains open until the program:

- Executes the CLOSE <cursor name> command.
- Returns control to the SQL prompt or the dot prompt. (Thus, the cursor remains open during execution of lower-level subroutines, whether these be dBASE program (.prg) files or SQL program (.prs) files.)
- Executes the SET SQL OFF command.
- Executes the CREATE DATABASE, START DATABASE, or STOP DATABASE command.

After the SELECT command is executed by OPEN, the system variable Sqlcnt contains the number of rows in the result table. If the result table is not empty (Sqlcnt contains a value other than 0), the FETCH command can be used for processing result rows.

NOTE
A declared cursor can be opened and closed as often as needed. A cursor must be closed before it can be reopened.

Example

To open cursor x_ptr after it has been DECLAREd, type:

```
DECLARE x_ptr CURSOR
 FOR SELECT *
 FROM Staff
 WHERE Hiredate > {9/01/86};
      .
      .
      .
 OPEN CURSOR x_ptr;
```

See Also

CLOSE, DECLARE CURSOR, FETCH

REVOKE

This command removes access privileges on tables and views previously granted using the GRANT command. The command format is similar to that for GRANT, and the privilege list can contain any privilege keyword allowed by GRANT.

Syntax

REVOKE ALL [PRIVILEGES]/<privileges list>
    ON [TABLE] <table list>
    FROM PUBLIC/<user list>;

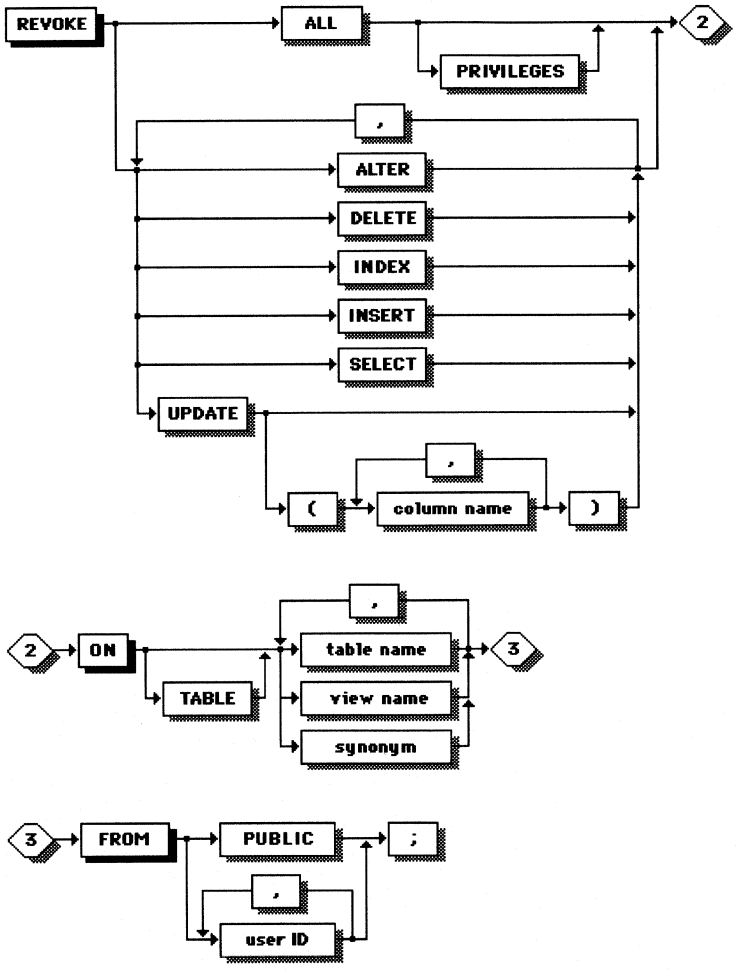


Figure 6-24 REVOKE command

Usage

The REVOKE command lets you remove one or more of the privileges that you have assigned using the GRANT command. You can use the REVOKE command to remove privileges:

- All at once, using the ALL keyword
- Individually using <privilege list>, which names privileges to be granted, separated by commas
- ON one or more tables or views using <table list>, which names tables and views on which privileges are to be granted, separated by commas
- FROM all users at once, using the PUBLIC keyword
- FROM individual users, using the <user list>, which specifies the user IDs to be denied privileges, separated by commas

The PRIVILEGES and TABLE keywords are used optionally for clarity.

You can only revoke privileges that you've previously granted (see note below). A grantee who has been granted the WITH GRANT OPTION can only revoke privileges that the grantee has granted to others using this option.

NOTE
A user cannot revoke privileges from the creator of a table or view or from the SQLDBA user ID. The SQLDBA user ID can revoke any privilege granted to a user, including one that the SQLDBA did not assign.

When you revoke privileges that you have granted to a grantee using WITH GRANT OPTION, these privileges are also revoked from other users to whom the grantee subsequently granted them. To revoke only the WITH GRANT OPTION for privileges previously granted to a user, revoke ALL privileges from the user and then use GRANT to grant them again without WITH GRANT OPTION.

If the same privilege has been granted to a user by more than one grantor, the privilege must be revoked by each grantor to be revoked completely. REVOKE...FROM PUBLIC revokes only those privileges granted to PUBLIC, not privileges individually granted.

Examples

To revoke privileges on the Staff table that were GRANTed to PUBLIC, type:

```
SQL. REVOKE ALL ON Staff
     FROM PUBLIC;
```

To revoke INSERT and UPDATE privileges on the Staff table from user ID John, type:

```
SQL. REVOKE INSERT, UPDATE ON Staff
     FROM John;
```

See Also

GRANT

ROLLBACK

This command restores table data altered during an aborted transaction.

Syntax

ROLLBACK [WORK];

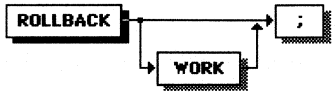


Figure 6-25 ROLLBACK command

Usage

You use the ROLLBACK command to undo changes made by SQL commands included within BEGIN TRANSACTION and END TRANSACTION commands. The WORK keyword can be included for clarity.

NOTE
You can include any SQL command in a transaction except the following:

- *Data definition commands (see the Classes of Commands section earlier in this chapter)*
- *Startup commands (START DATABASE and STOP DATABASE)*
- *Utility commands (see the Classes of Commands section earlier in this chapter)*
- *Database security commands (GRANT and REVOKE)*

Examples

Consider the following SQL program code:

```
ON ERROR DO Recover
SET REPROCESS TO 15
BEGIN TRANSACTION
   UPDATE Staff
   SET Commission = Commission * 1.5
   WHERE State = "NY";
END TRANSACTION
ON ERROR
IF COMPLETED()
   @ 21,15 SAY "Transaction successfully completed"
ENDIF
```

To restore the Staff table to its original state should an error occur during processing of the UPDATE command in the transaction, include the ROLLBACK command in the Recover error recovery procedure:

```
PROCEDURE Recover
   @ 21,15 SAY "Your transaction has encountered an error condition"
   @ 22,15 SAY "Do you want to RETRY? (Y/N)" GET choice PICTURE "!"
   READ
   @ 21,15 TO 22,65 CLEAR
   IF choice = "Y"
      RETRY
   ELSE
      @ 21,15 SAY "Rolling back your transaction. Please wait."
      ROLLBACK;
   ENDIF
RETURN
```

This command updates statistics in the SQL system catalog tables for a single table and its indexes, or for all tables and indexes in the current database.

Syntax

RUNSTATS [<table name>];

Figure 6-26 RUNSTATS command

Usage

Statistics are recorded in the SQL catalog tables during creation of tables and indexes. The statistics help SQL to determine the most efficient ways of performing database operations. As the database changes, statistics are not changed unless you periodically execute the RUNSTATS command to update them.

You can also use RUNSTATS to check for corrupted indexes and to compare system catalog entries with index tags in each multiple index file. An error message appears when any irregularity is detected.

Execute RUNSTATS with a <table name> after you've made the following types of changes to the table:

- Changed the table definition using the ALTER TABLE command
- Changed data in more than ten percent of the rows using INSERT, DELETE, and UPDATE commands
- Created new indexes using the CREATE INDEX command

Executing RUNSTATS without specifying a table name updates statistics for all tables and indexes in the current database. If the database file for any table in the current database is already open in dBASE mode when RUNSTATS is executed, you will receive a file-open error.

NOTE
Refer to Chapter 7 for more information about the SQL catalog tables and about the statistics updated using the RUNSTATS command.

Examples

To update statistics for all tables in the current database, type:

```
SQL. RUNSTATS;
```

To update statistics just for the Staff table, type:

```
SQL. RUNSTATS Staff;
```

This command retrieves data from tables or views for display in a result table, for use in another SELECT search condition, or for use by SQL program variables. You can execute SELECT commands interactively or embedded in SQL programs. If dBASE IV is password-protected, you must be the creator of, or have the SELECT privilege on, any table or view referenced in a SELECT command.

Syntax

SELECT <clause>
    [INTO <clause>]
    FROM <clause>
    [WHERE <clause>]
    [GROUP BY <clause>]
    [HAVING <clause>]
    [UNION <SELECT command>...]
    [ORDER BY <clause>/FOR UPDATE OF <clause>]
    [SAVE TO TEMP <clause>];

The following sections describe each clause of the SELECT command.

SELECT Clause

SELECT is a required clause. The SELECT clause specifies the columns, SQL aggregate functions, or expressions to be included in a SELECT command's result table. The expanded syntax of the SELECT clause is as follows:

SELECT [ALL/DISTINCT] <column list>/*

Every column whose name is specified in <column list> must exist in the tables or views named in the FROM clause. Column names must be separated from one another by commas.

Some of the columns in a column list may share the same name because they exist in different FROM tables and views. In this case, each name must be qualified by the name of its table or view.

Thus, for example, the Staff_no column of the Staff table is qualified as Staff.Staff_no, and the Staff_no column of the Sales table as Sales.Staff_no. To avoid having to qualify column names in a column list, you can specify alias names or synonyms for the tables and views in the FROM clause.

You can use an asterisk (*) in place of a column list to specify that all columns in FROM tables and views be included in the result. You also can qualify the asterisk with the name of a table from which you want to retrieve all columns, for example, Clients.*.

ALL is an optional keyword that specifies that all rows are to be displayed. If you do not use a WHERE clause in the SELECT command to limit the number of rows displayed, ALL rows are displayed by default.

DISTINCT is an optional keyword that eliminates all but one of each set of duplicate rows from the result table. When DISTINCT is used, the total concatenated length of <column list> cannot exceed 100 bytes. The DISTINCT keyword:

- Can be used only once in any SELECT clause of a SELECT command. This applies to the outer query and inner queries.
- Applies only to the first column named in a SELECT clause.

In addition to column names or the asterisk symbol, the SELECT clause can contain expressions and aggregate functions. An expression can be:

- A mathematical formula for specifying calculated column values (for example, *Salary*12*)
- A dBASE function that operates on column values (for example, *UPPER(City)*)
- A constant (for example, *''(Yearly Salary)''*) that can be used to describe the data in an adjacent result column
- A dBASE memory variable
- The USER keyword, which returns the user ID of the current user if dBASE IV is password-protected

An aggregate function consists of the keyword AVG, COUNT, MAX, MIN, or SUM, followed by a column name in parentheses, for example, *MAX(Salary)*. If a SELECT clause contains individual column names and aggregate functions, the SELECT command must contain a GROUP BY clause.

A heading is displayed for each column in a SELECT result table. A heading can be one of the following:

- For a column specified in the SELECT clause, the unqualified or qualified name of the column.
- For an expression specified in the SELECT clause, the prefix *EXP* and the left-to-right position number of the expression in the clause (*EXP1*, *EXP2*, and so on).
- For an aggregate function specified in the SELECT clause, the function name followed by its left-to-right position number in the clause (for example, *MAX1*, *MIN2*, *AVG3*).
- For a column specified in a GROUP BY clause, the prefix *G__* followed by the name of the column whose values are used to group result table values (for example, *G__DESCRIPT*).

NOTE

1. *To accommodate more column names in the SELECT clause, particularly when column names are qualified, substitute short synonyms or alias names for table names.*
2. *Before displaying lengthy result tables, specify SET PAUSE ON to display the information one screen at a time, or press* **Ctrl-S** *to halt and resume scrolling of data.*

INTO Clause

The INTO clause is an option used only in an embedded SQL program to process a single-row SELECT result. The expanded syntax of the INTO clause is as follows:

INTO <memvar1>[,<memvar2>,...]

For a discussion of how INTO operates, refer to Embedded Single-Row SELECT (with INTO) later in this section.

FROM Clause

FROM is a required clause that identifies the tables or views whose data is to be used in the result table. The expanded syntax of the FROM clause is as follows:

FROM <table name>/<view name> [alias] [,<table name>/<view name>[alias]...]

As an option, you can use the FROM clause to define alias names for the tables and views specified in the clause. An alias name is an alternate name that is substituted for the table or view name in a correlated subquery or self-join. An alias name can be up to ten characters long, must begin with a letter, and can contain letters, numbers, and underscores.

NOTE

You cannot use single letters A through J as alias names.

WHERE Clause

WHERE is an optional clause that specifies a search condition to qualify rows for inclusion in the result table. The expanded syntax of the WHERE clause is as follows:

WHERE [NOT]<search condition>
    [AND/OR [NOT] <search condition>...]

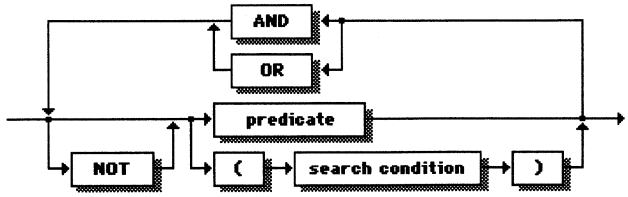


Figure 6-27 WHERE clause

To be included in the result table, any row in a table or view named in the FROM clause must satisfy the WHERE search condition. That is, the search condition must be true for the row. If the condition is false, the row is not included.

A search condition can be any valid expression, and can include column names, dBASE functions (but *not* aggregate functions), dBASE memory variables, operators, parentheses, and constants. For example, *Lastname = "Zambini"* is a simple search condition that selects only rows whose Lastname column values contain "Zambini". Figure 6-28 summarizes the use of SQL expressions.

You can use a SELECT command in a WHERE clause to supply values for the search condition, for example, *WHERE Salary > (SELECT AVG(Salary) from Staff)*.

You can use the following comparison operators in a search condition:

Operator	Description
<	Less than
>	Greater than
<=	Less than or equal to
>=	Greater than or equal to
=	Equals
<>, #, or !=	Not equal
!>	Not greater than
!<	Not less than

Comparisons in search conditions can only be made between expressions that have compatible data types. For example, a numeric column value must be compared with a numeric value, a character column value with a character value or character string, and so on. You can use dBASE functions such as VAL() and STR() to compare incompatible data types.

Character strings specified in conditions must be enclosed in matching single or double quotes.

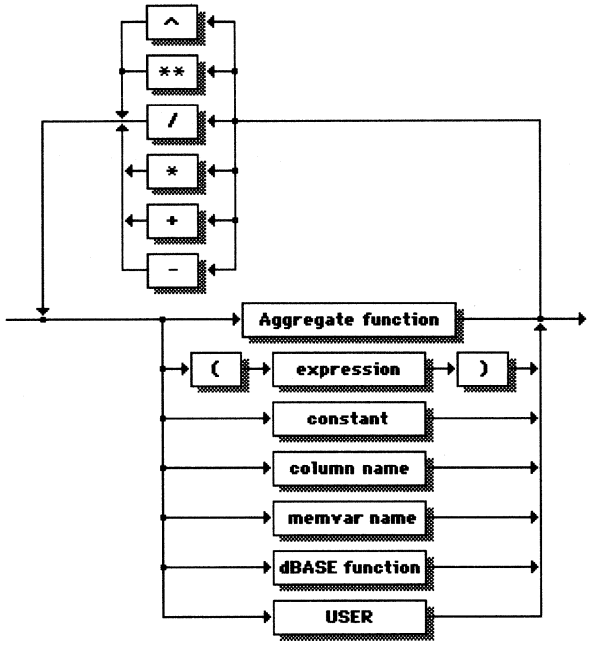


Figure 6-28 SQL expressions

You can use the logical operators AND, OR, and NOT to combine search conditions. For example, given the search condition *Lastname <> "Zambini" AND Age < 34*, a row would have to satisfy both conditions to be included in the result.

The order in which combined search conditions are evaluated is dictated by the precedence of the operators that connect them. A NOT condition is evaluated first, followed by AND and then OR conditions.

You may use parentheses to group conditions and change the order in which they are evaluated.

NOTE
Character-type values are compared using their ASCII values, which are evaluated from left to right. Because the ASCII values of upper-case letters are lower than the values of lowercase letters, the letter Z, *for example, is evaluated as less than the value of letter* a, *and* Z *does not equal* z.

WHERE search conditions also can include the predicates ALL, ANY, BETWEEN, EXISTS, IN, and LIKE. Predicates can be used in combination with:

- dBASE memory variables
- dBASE functions
- Subqueries
- The USER keyword (which returns the user ID of the current user)

Figure 6-29 summarizes the forms in which predicates can be used.

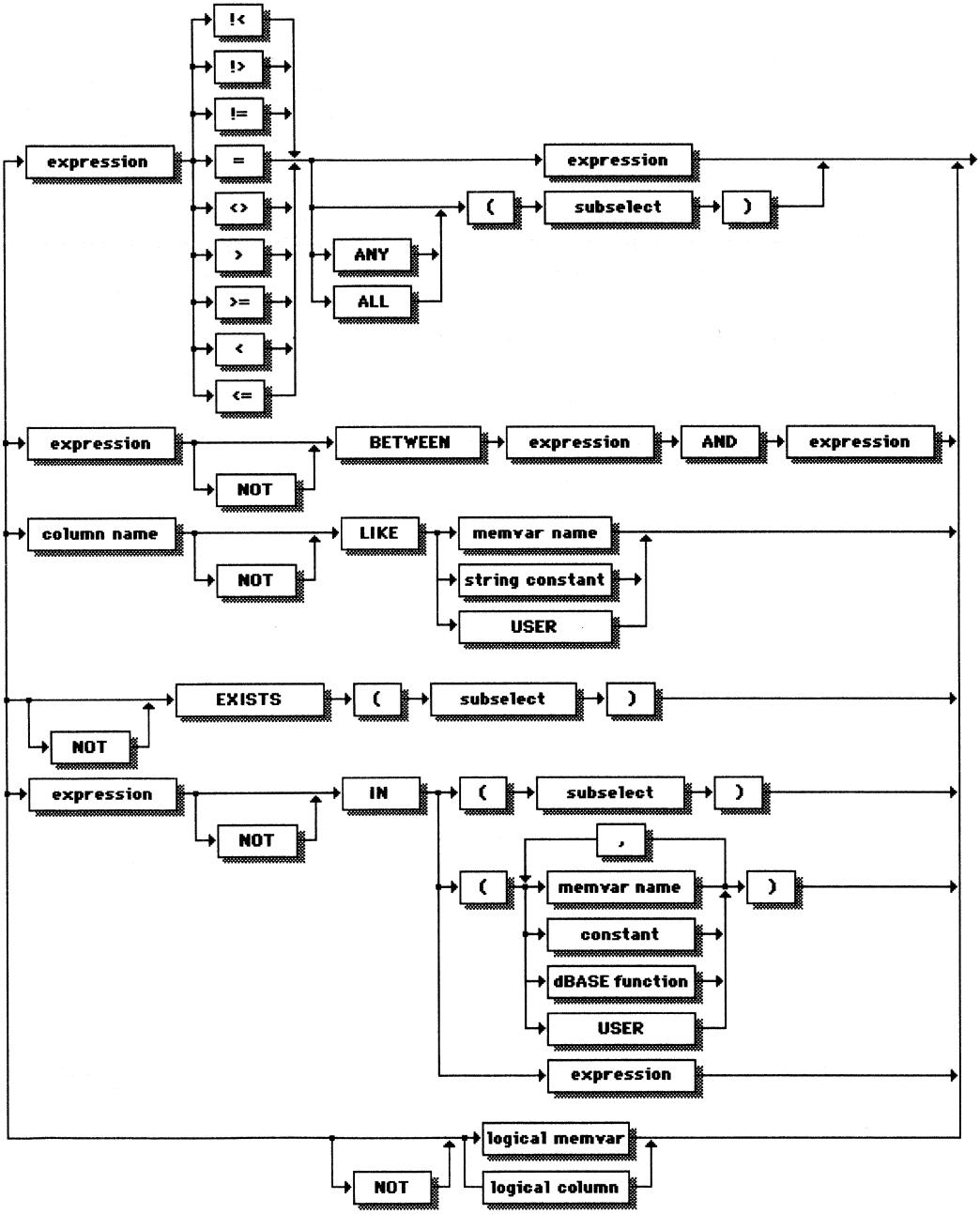


Figure 6-29 SQL predicates

GROUP BY Clause

GROUP BY is an optional clause that lets you summarize information for a group of rows that share a common value for a particular column. Each group is reduced to a single result row whose column values either describe the group or contain an aggregate group result. The expanded syntax for the GROUP BY clause is as follows:

GROUP BY <column name> [,<column name>...]

Each column named in a <column name> specification must also be specified in the SELECT clause. GROUP BY cannot specify a column whose values are derived using a mathematical or aggregate function. The total length of columns listed in GROUP BY cannot exceed 100 bytes.

You cannot use a GROUP BY clause in the following types of SELECT command:

- A SELECT command that contains an INTO clause
- A subquery whose FROM clause references a view whose definition contains a GROUP BY or HAVING clause

For each SELECT clause in a SELECT command (including that in the outer query and those in subqueries), there can be only one GROUP BY clause.

HAVING Clause

The HAVING clause is an optional clause used to specify a search condition that restricts the rows that appear in a group result. The expanded syntax of the HAVING clause is as follows:

HAVING [NOT]<search condition>
    [AND/OR [NOT] <search condition>...]

HAVING specifies one or more search conditions that evaluate to true or false for each row in the group. If the group is not defined using a GROUP BY clause, the entire result is treated as a single group. In this case, each column that is named in the SELECT clause normally is operated on by an aggregate function.

The following rules apply to the HAVING clause:

- Each HAVING search condition usually corresponds to an aggregate function named in the SELECT clause.
- The FROM clause of a SELECT query that is used in a HAVING search condition cannot name any table or view that is specified in the FROM clause in the main SELECT query.
- A correlated subquery cannot be used in a HAVING search condition.

UNION Clause

UNION is an optional clause that combines the result tables of two or more SELECT commands to produce a single result table without duplicate rows. The complete syntax of a query using the UNION clause is as follows:

```
<SELECT command> [UNION <SELECT command>...]
    [ORDER BY <clause>]
    [SAVE TO TEMP <clause> KEEP];
```

The following rules govern linking SELECT commands using UNION:

- Each column specified in the SELECT clause of one command must correspond to a column specified in the SELECT clause of another, though it need not have the same name. A union cannot be performed on a column with the LOGICAL data type.
- Corresponding columns must match in data type and length. Numeric columns must have the same number of digits; decimal numbers, the same number of digits to the right of the decimal point.
- A SELECT clause cannot contain dBASE memory variables or functions.

A union result is ordered by the values of all columns specified in each SELECT clause. You can specify a different order by placing an ORDER BY clause after the last SELECT command. Instead of a column name, you must specify a number indicating the left-to-right position of the column in the SELECT clause (for example, *ORDER BY 1*). (Refer to ORDER BY Clause, below.)

The SAVE TO TEMP clause lets you save a UNION result to a temporary table for use during the current session. The KEEP option allows you to store the result permanently as a dBASE database file. (Refer to SAVE TO TEMP clause, below.)

ORDER BY Clause

ORDER BY is an optional clause that allows you to specify the order of rows in a result table. You can specify ascending order (ASC, the default), descending order (DESC), or a combination of the two.

The expanded syntax of the ORDER BY clause is as follows:

```
[ORDER BY <column name>/<integer> [ASC/DESC]
    [,<column name>/<integer> [ASC/DESC]...];
```

Any column specified in an ORDER BY clause must also be specified in the SELECT clause. The total concatenated length of columns listed in the ORDER BY clause cannot exceed 100 bytes.

You can specify an ORDER BY column by name or by a number that indicates its left-to-right position in the SELECT clause. Use of a number is necessary to enable you to order a result by values derived using constants, dBASE functions or memory variables, calculations on column values, and other expressions.

If you specify more than one column in an ORDER BY clause, result table rows are first arranged according to the values in the first column specified. Rows that contain the same first-column value are then arranged according to the values in the second column, and so on. For each ORDER BY column, you can specify ascending or descending order.

FOR UPDATE OF Clause

FOR UPDATE OF is an optional clause used in a SELECT command defined using the DECLARE CURSOR command. Therefore, FOR UPDATE OF can only be used in an embedded SQL program.

FOR UPDATE OF specifies the columns that can be updated by a subsequent UPDATE command that contains a WHERE CURRENT OF clause. The complete syntax of the FOR UPDATE OF clause is as follows:

FOR UPDATE OF <column name> [,<column name>...]

The FOR UPDATE OF columns must be contained in a table or updatable view named in the FROM clause of the SELECT command. You cannot use the FOR UPDATE OF clause in a SELECT command that also contains an INTO, ORDER BY, or SAVE TO TEMP clause.

SAVE TO TEMP Clause

SAVE TO TEMP is an optional clause that saves the SELECT result either as a temporary table that can be used only during the current session or as a permanent dBASE database file. The expanded syntax of the SAVE TO TEMP clause is as follows:

SAVE TO TEMP <table name> [(<column name> [,<column name>...])]
    [KEEP]

SAVE TO TEMP saves a result in a temporary table named using the <table name> specification. The same rules for naming permanent tables (refer to CREATE TABLE) also apply to naming temporary tables. Column data types and lengths are the same as defined for the base table or view.

The optional <column name> specification is used in the following ways:

- To assign names to columns whose data is derived using an expression, dBASE or aggregate function, memory variable, or constant. (For each of these types of columns, you must use a <column name> specification.)
- To change the names of the columns as defined in the base table or view.

If you omit the KEEP option, the result table that you create is temporary and can be used:

- In interactive SQL mode, only during the remainder of your SQL session with the same database. Once you switch to a different database or return to the dBASE dot prompt, the table is dropped.
- In embedded SQL mode, only until the program returns control to the dot prompt or SQL prompt.

If you use the KEEP option, the temporary result table is saved as a permanent, unencrypted dBASE database file in the current directory. This file can then be used in dBASE IV (for example, to produce a report), or you can use the file to create a new SQL table using the DBDEFINE commmand.

NOTE
You cannot use the SAVE TO TEMP clause if the SELECT command contains a FOR UPDATE OF or INTO clause.

If SET CATALOG is ON, SAVE TO TEMP adds an entry for the temporary table to the open dBASE catalog file and removes the entry when the table is later erased. If the KEEP option is used, the entry for the temporary table is replaced by an entry for the new database file.

Usage

SELECT is the most versatile SQL command. The command can be entered by itself or used as a subquery in such commands as INSERT, UPDATE, and DELETE.

The SELECT syntax changes, depending on how the command is used:

- Display-only SELECT
- Embedded single-row SELECT
- Embedded SELECT used with DECLARE CURSOR

Display-Only SELECT

You can use the following SELECT syntax in both embedded and interactive SQL modes to retrieve and display table and view data. The expanded SELECT syntax for retrieval operations is as follows:

```
SELECT [ALL/DISTINCT] <column list>
    FROM <table name>/<view name> [alias]
        [,<table name>/<view name> [alias]...]
    [WHERE [NOT]<search condition>
        [AND/OR [NOT] <search condition>...]]
    [GROUP BY <column name> [,<column name>...]]
    [HAVING [NOT]<search condition >
        [AND/OR [NOT] <search condition>...]]
    [ORDER BY <column name>/<integer> [ASC/DESC]
        [,<column name>/<integer> [ASC/DESC]...]]
    [SAVE TO TEMP <table name>
        [(<column name> [,<column name>...]) [KEEP]];
```

When the UNION clause is used for retrieval, the syntax is as follows:

```
<SELECT command> [UNION <SELECT command>...]
    [ORDER BY <column name>/<integer> [ASC/DESC]
        [,<column name>/<integer> [ASC/DESC]...]]
    [SAVE TO TEMP <table name>
        (<column name> [,<column name>...]) [KEEP]];
```

NOTE
You can include the FOR UPDATE OF clause in an interactive SELECT command to test the results of a query before using it in an embedded SQL program. You cannot use a FOR UPDATE OF clause in a SELECT command that contains an ORDER BY clause.

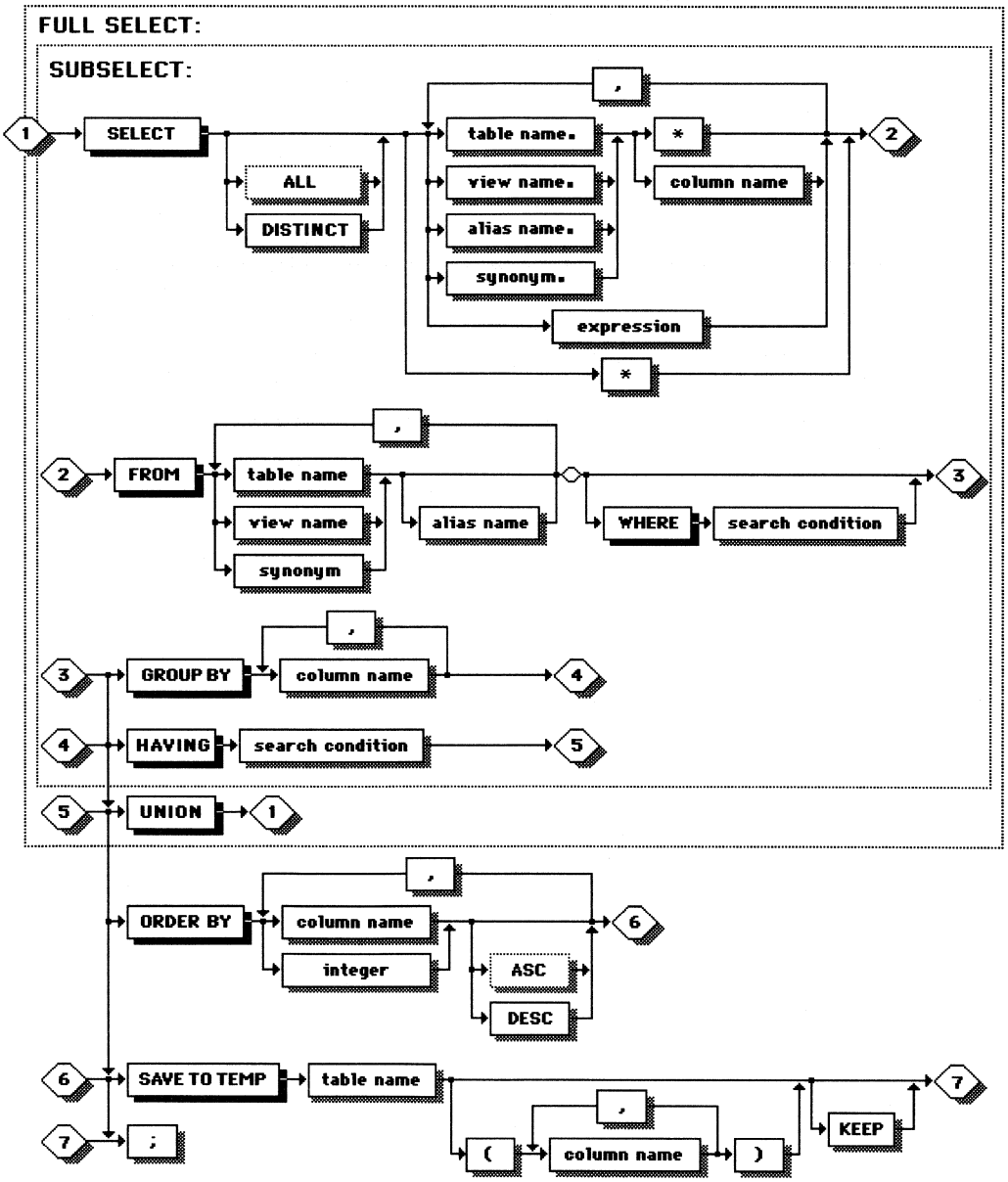


Figure 6-30 Display-only SELECT

You can use a SELECT command to *join* information from several tables. A join allows you to display data from more than one table without writing a program. In the SELECT command that defines a join, you can:

- Specify the columns in each table that you want to appear in the result.
- Use the asterisk (*) symbol to display all columns from all the joined tables.
- Use the WHERE clause to specify *join conditions* that limit the rows that appear in the result.
- Use GROUP BY and HAVING to group the result and ORDER BY to order it.

A SELECT command also can be *nested* within the WHERE clause or HAVING clause of another SELECT command. This type of command, called a *subquery* or *inner* query, supplies values for the search condition of the *outer* query.

There are two types of SELECT subquery command, *simple* and *correlated*:

- Simple — The inner query is evaluated first and its result used to evaluate the outer query.
- Correlated — The outer query is evaluated first and its result used to evaluate the inner query. The inner query is evaluated once for each row returned by the outer query.

Besides using the SELECT command to display data, you can use SELECT as a subquery in the following SQL commands:

- DELETE and UPDATE: As part of a WHERE search condition
- INSERT: To select rows for insertion in a table
- SELECT: As part of a WHERE or HAVING search condition
- CREATE VIEW: To define the view

Used as a subquery, a SELECT command can contain FROM, WHERE, GROUP BY, and HAVING clauses.

Embedded Single-Row SELECT (with INTO)

A single-row SELECT command returns one result row. In embedded SQL mode, you can use a single-row SELECT with the INTO clause to transfer result row values into dBASE memory variables. The expanded syntax of this form of the SELECT command is as follows:

```
SELECT [ALL/DISTINCT] <column list>
    INTO <memvar list>
    FROM <table name>/<view name> [alias]
        [,<table name>/<view name> [alias]...]
    [WHERE [NOT]<search condition>
        [AND/OR [NOT] <search condition>...]];
```

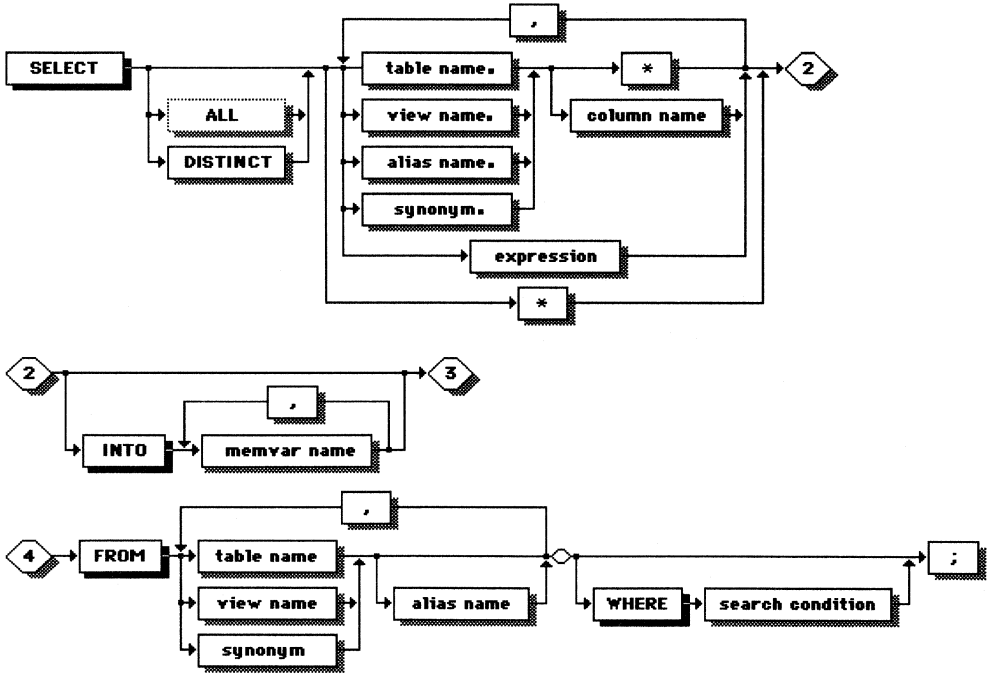


Figure 6-31 Embedded single-row SELECT (with INTO)

NOTE
You cannot use INTO in a SELECT command that contains a FOR UPDATE OF, GROUP BY, HAVING, UNION, ORDER BY, or SAVE TO TEMP clause. If a SELECT command returns more than a single row, only the values for the first result row are used for INTO memory variables.

Values from the single result row are transferred into the variables listed in the INTO clause one for one, from left to right. If the INTO variables do not already exist, they are created during transfer using the data types of their corresponding row values.

Embedded SELECT (with DECLARE CURSOR)

In an embedded SQL program, you use a SELECT command defined in a DECLARE CURSOR command to produce a result table. The expanded syntax of a cursor's SELECT command is as follows:

```
SELECT [ALL/DISTINCT] <column list>/*
    FROM <table name>/<view name> [alias]
        [,<table name>/<view name> [alias]...]
    [WHERE [NOT]<search condition>
        [AND/OR [NOT] <search condition>...]]
    [GROUP BY <column name> [,<column name>...]]
    [HAVING [NOT]<search condition>
        [AND/OR [NOT] <search condition>...]]
    [ORDER BY <column name>/<integer> [ASC/DESC]
        [,<column name>/<integer> [ASC/DESC]...]]/
    [FOR UPDATE OF <column name> [,<column name>...]];
```

After the OPEN command activates the cursor and executes its SELECT command, a FETCH command is used to point the cursor to each row of the SELECT result in turn, and transfer row values into dBASE memory variables to enable one of the following operations:

- An UPDATE of column values for the row in the base table that corresponds to the pointed-to result row
- A DELETE of the row in the base table that corresponds to the pointed-to result row

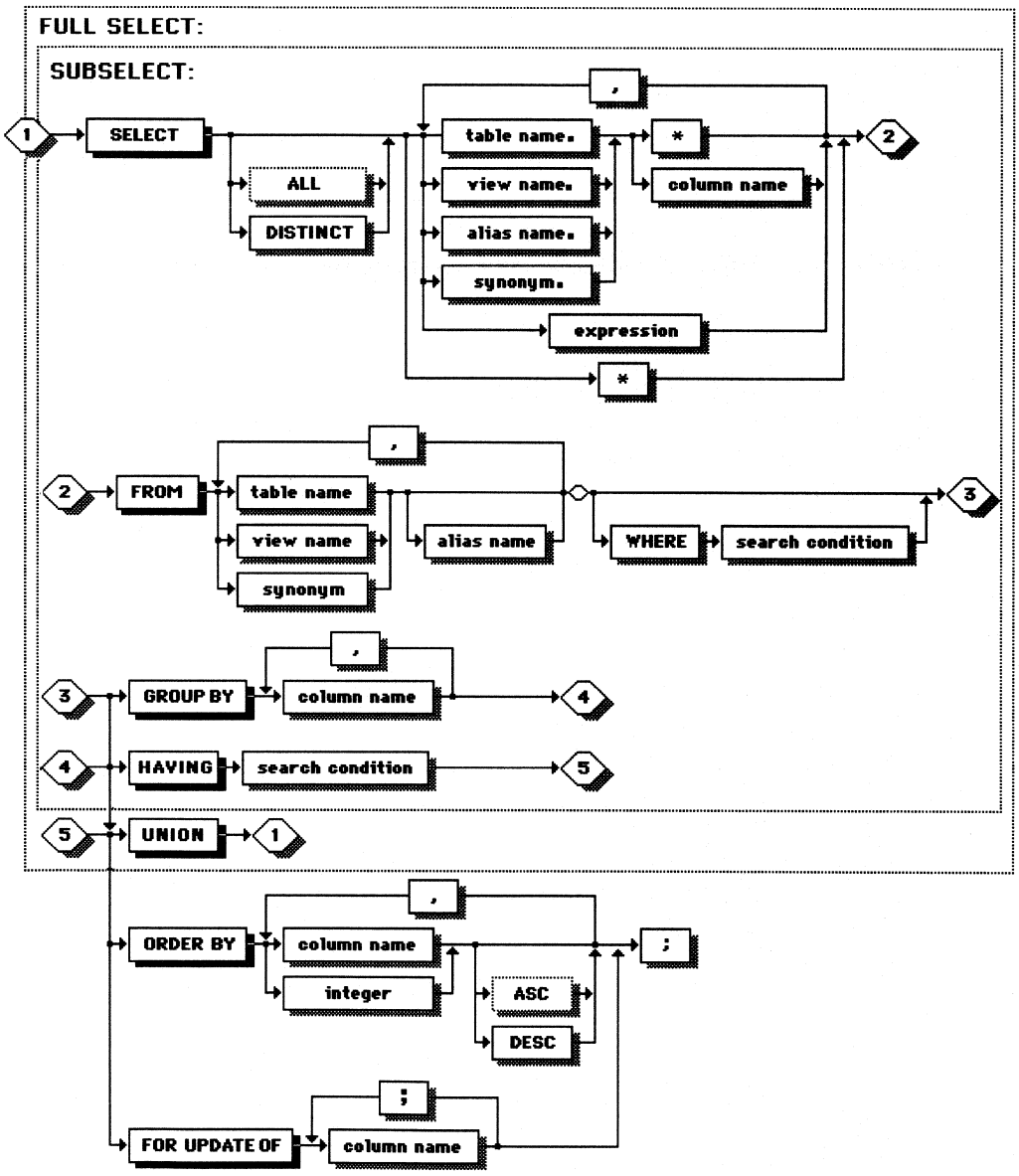


Figure 6-32 Embedded SELECT (with DECLARE CURSOR)

Examples

To display all columns of the Sales table, type:

```
SQL. SELECT *
     FROM Sales;
```

To display specific columns from the Customer table, type:

```
SQL. SELECT Company, Firstname, Lastname
     FROM Customer;
```

To display only unique row values for specific columns of the Inventry table, type:

```
SQL. SELECT DISTINCT Part_no, Descript, Unitcost
     FROM Inventry;
```

To display information only for selected rows in the Inventry table, type:

```
SQL. SELECT Part_no, Descript, Location, On_hand
     FROM Inventry
     WHERE Location = "CHICAGO";
```

To display Inventry row values that satisfy several search conditions (grouped by parentheses), type:

```
SQL. SELECT Part_no, Descript, Location, On_hand, Unitcost
     FROM Inventry
     WHERE (Location = "LOS ANGELES" OR Location = "NEW YORK")
     AND (On_hand < 20 AND Unitcost < 1000);
```

To combine ordinary Staff row values with calculated row values and a value produced by a constant expression, type:

```
SQL. SELECT Staff_no, Firstname, Lastname,
     Salary*12, "(Yearly Salary)"
     FROM Staff;
```

To use the dBASE DATE() function in a WHERE clause search condition to select Staff rows, type:

```
SQL. SELECT Firstname, Lastname, Hiredate
     FROM Staff
     WHERE (DATE()-Hiredate) > 365*5;
```

To order Inventry result rows using an ORDER BY clause, type:

```
SQL. SELECT Part_no, Descript, Location, On_hand
     FROM Inventry
     ORDER BY Descript ASC;
```

To use an aggregate function to produce a result from the Customer table, type:

```
SQL. SELECT COUNT(*)
     FROM Customer
     WHERE State = "NY";
```

To group Inventry result rows by part number using GROUP BY, type:

```
SQL. SELECT Part_no, SUM(On_hand)
     FROM Inventry
     GROUP BY Part_no;
```

To combine data from the Sales and Staff tables using a join, type:

```
SQL. SELECT Order_no, Sale_date, Staff.Staff_no, Lastname
     FROM Sales, Staff
     WHERE Sales.Staff_no = Staff.Staff_no;
```

To insert data into a new table using a SELECT command, type:

```
SQL. INSERT INTO LA
     SELECT *
     FROM Staff
     WHERE Lastname = "LOS ANGELES";
```

See Also

DECLARE CURSOR, DELETE, FETCH, INSERT, UPDATE

SHOW DATABASE

This command displays information about each SQL database. This command can be executed without a database in use.

Syntax

SHOW DATABASE;

Figure 6-33 SHOW DATABASE command

Usage

The SHOW DATABASE command displays the following information about each database:

- Database name
- User ID of the creator of the database (on password-protected systems)
- Date the database was created
- DOS path of the database directory

This information is stored in the master catalog table, Sysdbs, in the dBASE IV SQL *home* directory (SQLHOME).

NOTE
The SQLHOME= *setting in Config.db specifies the DOS path for the SQL home directory.* SQLHOME=C:\DBASE\SQLHOME *is the default.*

Example

To display information about all available databases, type:

```
SQL. SHOW DATABASE;
```

See Also

CREATE DATABASE, DROP DATABASE, START DATABASE, STOP DATABASE

START DATABASE

This command activates a database to which all subsequently executed SQL commands are to apply.

Syntax

START DATABASE [<database name>];

Figure 6-34 START DATABASE command

Usage

The START DATABASE command is normally the first command that you execute before entering SQL commands interactively. Often, START DATABASE is also the first command executed in a SQL program file.

Only one database can be active at a time. To stop one database and start another, execute START DATABASE for the new database. The preceding database is automatically closed and all cursors and temporary tables associated with it are dropped. To stop the current database without starting another, execute the STOP DATABASE command.

If SET CATALOG is ON, START DATABASE adds all the tables in the STARTed database to the open dBASE catalog file. It does not add SQL system catalogs to the open catalog file.

You can activate a database without executing START DATABASE by specifying a default database in your Config.db file using the SQLDATABASE = <database name> command. This default database is started automatically when you enter SQL mode by executing the SET SQL ON command.

If you execute START DATABASE without specifying a database name, or if a database that you specify cannot be opened, START DATABASE displays an error message and opens your default database. When you create a database using the CREATE DATABASE command, the new database is automatically activated.

You can include the START DATABASE command without a database name in a SQL application, provided that:

- START DATABASE is the first command to be executed by the application.
- The contents of the database directory where the application was developed, including catalog tables, have been copied to the directory from which you start dBASE IV (the current directory).

START DATABASE

NOTE
Such an application is called a ''single-database'' application. It is considered ''portable'' because it can be moved to a computer other than the one on which it was developed and run without compiling.

See Also

CREATE DATABASE, DROP DATABASE, SHOW DATABASE, STOP DATABASE

STOP DATABASE

This command closes the current database.

Syntax

STOP DATABASE;

Figure 6-35 STOP DATABASE command

Usage

A database must be closed before it can be dropped using the DROP DATABASE command. You can close the current database in one of two ways:

1. Execute the STOP DATABASE command
2. Execute the START DATABASE command to activate a new database

When a database is closed, all cursors and temporary tables associated with it are dropped.

See Also

CREATE DATABASE, DROP DATABASE, SHOW DATABASE, START DATABASE

UNLOAD DATA

This command exports data from a SQL table to an external file. If dBASE IV is password-protected, you must be the creator of the table from which data is UNLOADed or have the SELECT privilege on the table.

Syntax

UNLOAD DATA TO [path]<filename>
FROM TABLE <table name>
[[TYPE] SDF/DIF/WKS/SYLK/FW2/RPD/DBASEII/
DELIMITED [WITH BLANK/WITH <delimiter>]];

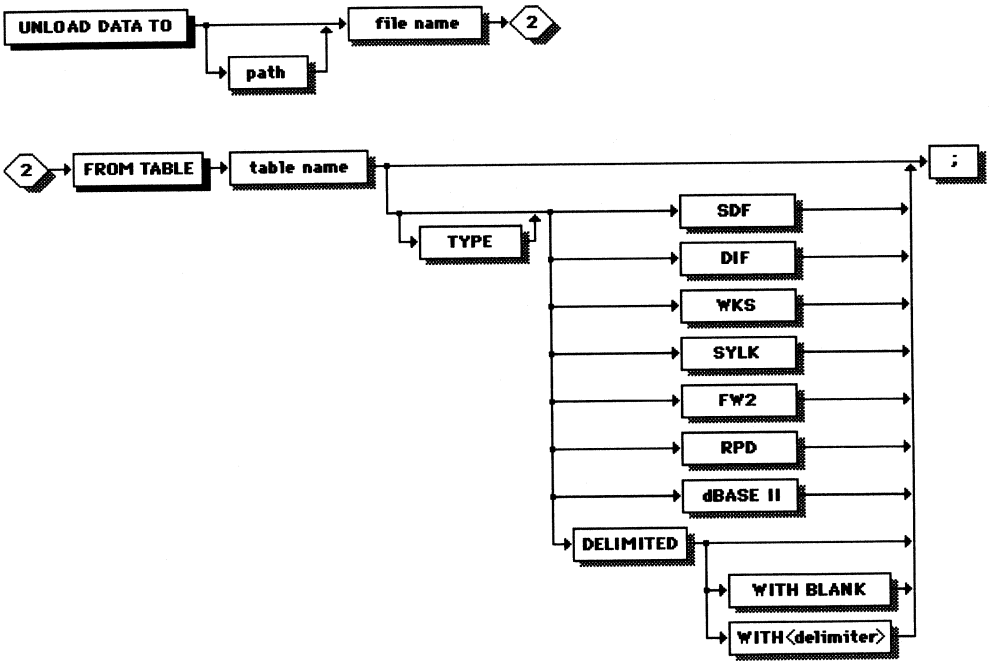


Figure 6-36 UNLOAD DATA command

Usage

The UNLOAD DATA command creates a new file and copies data to it from a SQL table. The UNLOAD DATA command supports the same file formats as the dBASE COPY TO command. You can export data to the following types of files:

- dBASE II database files (dBASEII)
- dBASE III, dBASE III PLUS, and dBASE IV database files
- RapidFile database files (RPD)
- Framework II database and spreadsheet files (FW2)
- Delimited format ASCII files (DELIMITED)
- System data format ASCII files (SDF)
- VisiCalc format files (DIF)
- Multiplan spreadsheet format files (SYLK)
- Lotus 1-2-3 format files (WKS)

UNLOAD DATA creates the export file in the current user directory (not the current database directory) unless you specify an explicit path. If you do not specify a file TYPE, data is exported to a dBASE database file. Regardless of whether dBASE IV is password-protected, the command always exports data to an unencrypted file.

Fields in the export file are created using the column names and definitions of the copied table data. For spreadsheet files, table column names are used as column headers.

The DELIMITED option specifies export to a delimited format ASCII text (.txt) file. The UNLOAD DATA command makes the following assumptions about the file to which you wish to export based on your DELIMITED specification:

- DELIMITED — File fields are separated by commas and character fields are delimited by double quotes. This can also be expressed by DELIMITED WITH ".
- DELIMITED WITH BLANK — File fields are separated by one space and character fields are not delimited.
- DELIMITED WITH <delimiter> — File fields are separated by commas and character fields are delimited by the specified delimiter. Double quotes is the default delimiter.

UNLOAD DATA

Example

To export data from the Staff table to a dBASE database file in the current user directory, type:

```
SQL. UNLOAD DATA TO C:\BACKUP\Staff2 FROM TABLE Staff;
```

See Also

LOAD DATA

This command changes column values within specified rows of a table or updatable view (refer to CREATE VIEW).

There are two forms of the UPDATE command. The first form can be used in both interactive and embedded SQL modes. The second, used only in embedded SQL mode with a declared cursor, updates the row in the base table that corresponds to the row in the result table that is pointed to by the cursor.

Syntax

UPDATE <table name>/<view name> (1)
SET <column name> = <expression>
[,<column name> = <expression> ...]
[WHERE <search condition>];

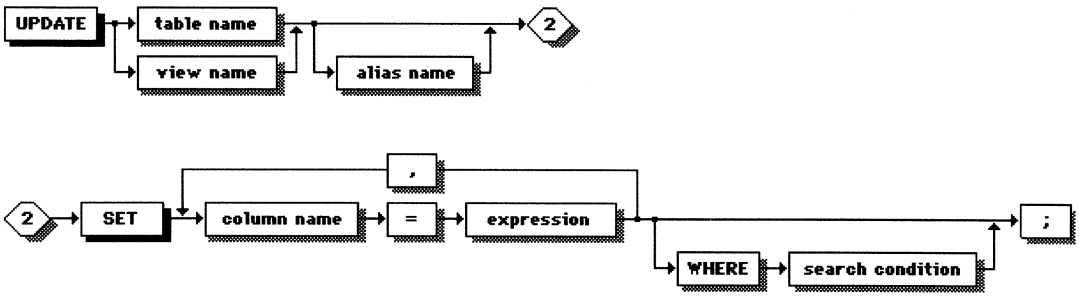


Figure 6-37 UPDATE command

UPDATE <table name> (2)
SET <column name> = <expression>
[,<column name> = <expression>...]
WHERE CURRENT OF <cursor name>;

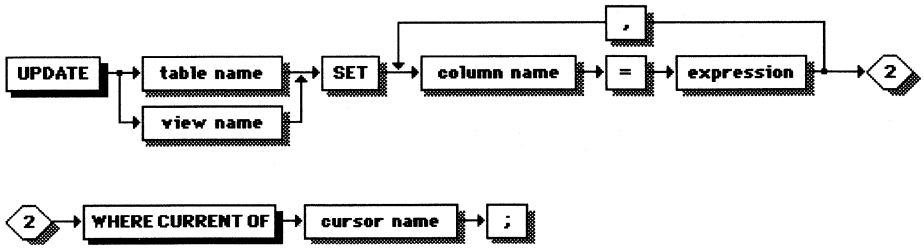


Figure 6-38 UPDATE command with WHERE CURRENT OF Clause

Usage

If dBASE IV is password-protected, you must be the creator of the table or view to be updated or have the UPDATE privilege for each column that you want to update.

UPDATE with WHERE Clause

The WHERE clause allows you to select the rows that you want to update. WHERE can contain a search condition or a subquery. A subquery used in the WHERE clause cannot also reference the table or view that you are updating.

NOTE
When you update a view defined with WITH CHECK OPTION, the updated column values must meet the conditions specified in the WHERE clause that defines the view.

UPDATE with WHERE CURRENT OF Clause

You use the WHERE CURRENT OF clause in an embedded SQL program to update base table rows that correspond to result table rows pointed to by a cursor. The cursor referenced in WHERE CURRENT OF must have been defined by a DECLARE CURSOR command earlier in the program and then opened by an OPEN command.

The base table row that is updated corresponds to the result row to which the cursor is currently pointing, that is, the row whose values were most recently FETCHed.

TIP
To verify that the WHERE clause of an UPDATE command will update the correct rows, first test the clause in a SELECT command.

Examples

To update the Commission column of the Staff table to reflect an increase in commissions for tenured salespeople, type:

```
SQL. UPDATE Staff
     SET Commission = Commission * 1.25
     WHERE Hiredate < {07/05/82};
```

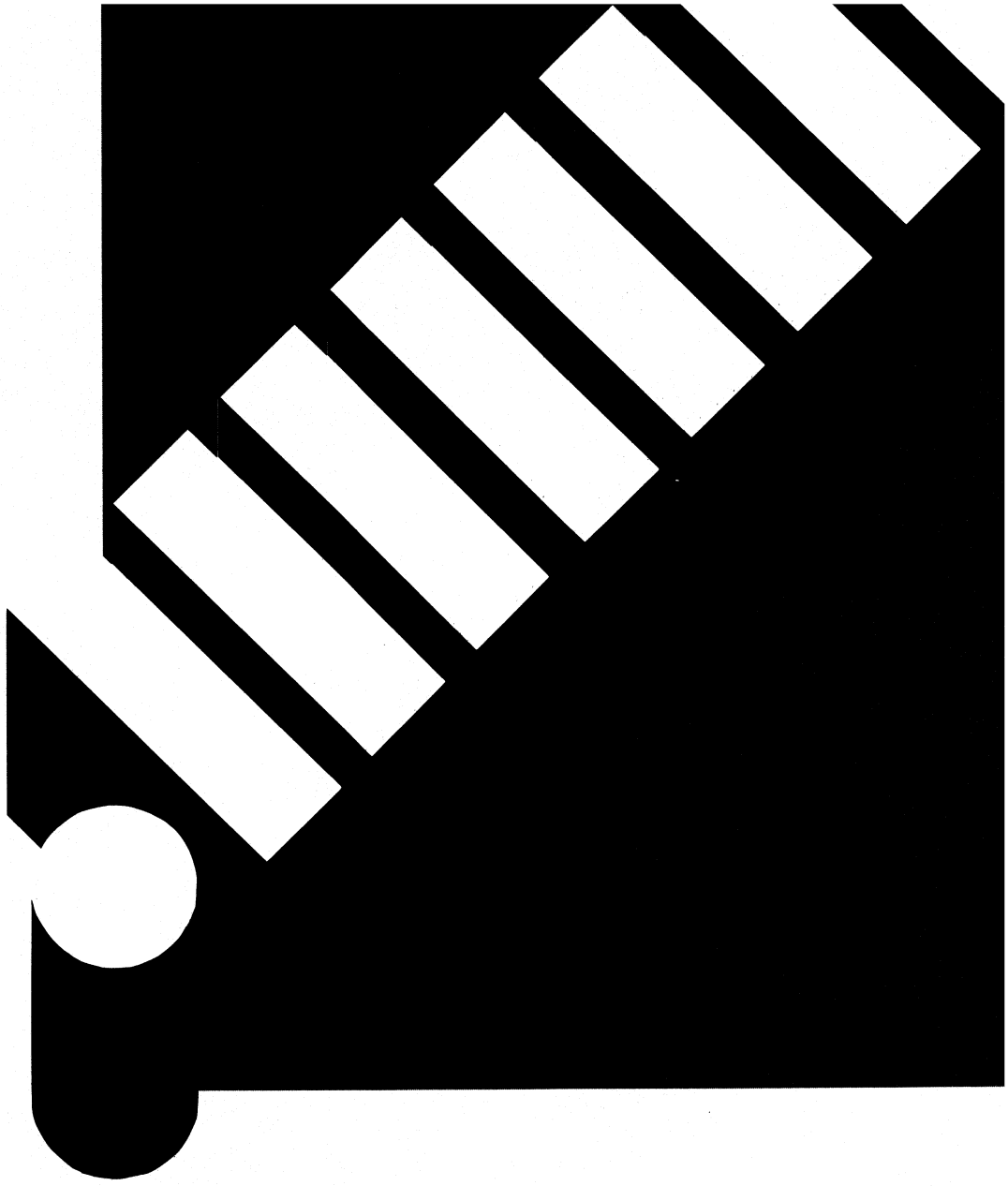
Use the following program segment to update the Sales table to reflect invoiced orders:

```
DECLARE Inv CURSOR FOR
 SELECT Order_no, Sale_date, Staff_no, Cust_no, Invoiced
 FROM Sales
 WHERE NOT Invoiced
 FOR UPDATE OF Invoiced;
   .
   .
   .
OPEN Inv;
 DO WHILE .T.
   FETCH Inv INTO morder_no, msale_date, mstaff_no, mcust_no, minvoiced;
   IF SQLCODE = 0
   * Print an invoice for order number in the the current row
   * If successful, change minvoiced memory variable to .T.
   DO Invoice WITH morder_no, msale_date, mstaff_no, mcust_no, minvoiced
     IF minvoiced
        UPDATE Sales
        SET Invoiced = .T.
        WHERE CURRENT OF Inv;
     ENDIF
   ELSE
      EXIT
   ENDIF
 ENDDO
CLOSE Inv;
```

In this example, the Invoice program generates an invoice for the order row pointed to by the cursor. After the order is invoiced, the Invoiced value in the current row is updated to .T. (True).

See Also

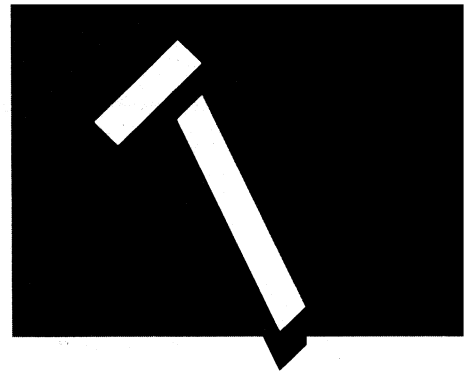
DECLARE CURSOR, FETCH, OPEN, SELECT



i 1 2 3 4 5 6 7 A B

C D E F G H In

SQL Catalogs



dBASE IV SQL keeps track of all *objects* (tables, views, synonyms, indexes) in a database using *catalog tables*. dBASE IV SQL uses the information in the catalog tables when performing queries or updates.

A set of catalog tables is constructed for each database you create with the CREATE DATABASE command. A master catalog table, Sysdbs, is maintained in the SQL *home* directory (default is \DBASE\SQLHOME) to keep track of each SQL database. The Sysdbs table contains the name of each database, the user ID of the person who created the database, the date it was created, and the full operating system path to the database. You can use the SHOW DATABASE command to list the databases in the Sysdbs table.

The catalog for each database consists of 10 SQL tables described below.

Table Name	Description
Sysauth	Describes the privileges held by users on tables and views — one row is entered for each user for each table or view for which a successful grant of one or more privileges has been made
Syscolau	Describes the privileges held by users to update columns in a table or updatable view — one row is entered for each GRANT operation for each grantee for each column for which the UPDATE privilege was granted
Syscols	Describes each column in every table and view, including the catalog tables — one row per column (for every table or view in the current database)
Sysidxs	Describes every index in the database and the table for which it's defined — one row per index
Syskeys	Describes every column (index key) used in each index — one row for each column for every index in the current database
Syssyns	Contains all synonym definitions for each table and view — one row for every synonym defined in the current database
Systabls	Contains information describing every table and view — one row for every table or view defined in the current database

(continued)

Table Name	Description
Systimes	Contains information used in multi-user environments to ensure that users have the latest copies of SQL catalog tables
Sysvdeps	Describes the relation between views and tables in the current database (so that if tables are dropped, views based on those tables will also be dropped) — one row for each table used in a view definition in the current database
Sysviews	Contains view definitions and the limitations on use of each view — one or more rows for each view definition

NOTE
The catalog tables that SQL uses are similar to those defined for the IBM DB2 database product, with variations due to the differences between the mainframe and microcomputer environments.

You can display information from a system catalog table using the SELECT command, just as you can any other SQL table. For example, to display all of the tables and views in the current database as stored in the Systabls table, you could type:

```
SQL. SELECT Tbname, Creator, Tbtype
       FROM Systabls;
```

This SELECT command displays information for both tables and views in the current database. The Tbtype column indicates whether files are tables (T) or views (V). Letters D and K specify temporary tables created with the SELECT...SAVE TO TEMP command. (K indicates that the KEEP option was specified.)

Although you can view the contents of the catalog tables, you cannot update them.

NOTE
The SQLDBA user ID has special privileges, and can directly update catalog tables with the UPDATE command. The SQLDBA user ID cannot, however, use the ALTER command to change the structure of the SQL catalog tables.

Updating the Catalog Tables

The catalog tables contain two types of information, SQL object definitions and statistics related to stored data. SQL object definitions specify information about SQL objects, databases, tables, views, synonyms, and indexes and are automatically updated by SQL commands. When you create or add objects to a database, or add or change table and view authorization privileges, the SQL catalog tables are automatically updated. Catalog tables are also updated for specific tables when you CREATE INDEXes or execute the DBDEFINE command.

Catalog tables are not automatically updated for changes to data in tables, for example, those made with INSERT, UPDATE, or DELETE. Because SQL uses statistical information on tables and associated indexes to optimize its performance, you need to update the catalog tables periodically with the RUNSTATS utility command.

Normally, you should use the RUNSTATS command when you've made changes to a table definition, just DBDEFINEd a .dbf file with an associated production .mdx file, or added or changed more than 10% of the rows in a table. You should not perform any other database operation while using the RUNSTATS command. For information about executing RUNSTATS, refer to Chapter 6.

RUNSTATS checks the catalogs against the actual database tables and views and any other objects in the database, and updates the catalog tables for discrepancies. After RUNSTATS has finished, the message **RUNSTATS successful** displays if all updates were made. If a problem occurred, the message **RUNSTATS completed without catalog updates. Error/warning found** appears. If this message appears, follow the suggested remedies for messages preceding the RUNSTATS message and execute RUNSTATS again.

The columns that RUNSTATS updates are the following:

Column	Table
COLCARD	Syscols
HIGH2KEY	Syscols
LOW2KEY	Syscols
FIRSTKCARD	Sysdxs
FULLKCARD	Sysidxs
NLEAF	Sysidxs
NLEVEL	Sysidxs
UPDATED	Systabls
CARD	Systabls
NPAGES	Systabls

NOTE
COLCARD, HIGH2KEY, and LOW2KEY are updated only for columns of numeric data type (Decimal, Float, Integer, Numeric, or Smallint).

Description of Catalog Tables

The following sections describe each SQL catalog table and the information stored in each column of the catalog table.

Sysauth Table

The Sysauth table contains information about the privileges held by users on tables and views. This table is updated by the GRANT and REVOKE commands. One row of 77 bytes is added for every assignment of privileges to a user (IDs assigned by PROTECT) for each table in the current database. The number of rows generated by each successful GRANT is the product of the number of grantees specified in the TO list and the number of tables specified in the ON list.

Column	Type	Description
GRANTOR	Char(10)	User ID granting these privileges.
TNAME	Char(10)	Table to which privileges apply.
USERID	Char(10)	User ID of person to whom privileges are granted.
TBTYPE	Char(1)	T for tables and V for views. D and K specify temporary tables created with the SELECT... SAVE TO TEMP command (K indicates that the KEEP option was specified).
PSELECT	Numeric(8)	Integer value time stamp of the date when the privilege was granted.
PINSERT	Numeric(8)	Integer value time stamp of the date when the privilege was granted.
PDELETE	Numeric(8)	Integer value time stamp of the date when the privilege was granted.
PALTER	Numeric(8)	Integer value time stamp of the date when the privilege was granted.
PINDEX	Numeric(8)	Integer value time stamp of the date when the privilege was granted.
PUPDATE	Char(4)	Scope of privileges granted for UPDATE: ALL (all columns), SOME (specified columns), or NONE (if no privileges granted).
GRANTOPT	Char(1)	Set to G if user is given GRANT privileges (allowing user to grant privileges to other users); otherwise set to Y.

Syscolau Table

The Syscolau table contains information about each user's UPDATE privileges for each column in a table or view in the current database. This catalog table is updated by the GRANT and REVOKE commands. One row of 50 bytes is added for each GRANT operation for each grantee (user ID or PUBLIC) and every column for which UPDATE privileges are granted. The number of rows generated by each successful GRANT is the product of the number of grantees specified in the TO list and the number of columns for which the UPDATE privilege is granted.

Column	Type	Description
GRANTOR	Char(10)	User ID of person assigning UPDATE privileges (or creator of view)
TNAME	Char(10)	Table or view to which privilege applies
GRANTEE	Char(10)	User ID to whom privilege is granted
COLNAME	Char(10)	Name of column for which UPDATE privilege is granted
AUTHTIME	Numeric(8)	Integer value time stamp of the date when the privilege was granted
GRANTOPT	Char(1)	Set to G if user is given GRANT privileges (allowing user to grant privileges to other users); otherwise set to Y

Syscols Table

The Syscols table contains information that describes each column in every table and view in the current database. One row of 68 bytes is added for each column defined in each table or view in the database (including all columns in the catalog tables). New rows are automatically added when new tables or views are added, or when existing tables are ALTERed to include additional columns.

Column	Type	Description
COLNAME	Char(10)	Column name
TBNAME	Char(10)	Table or view in which the column is defined
TBCREATOR	Char(10)	User ID of person who created the table
COLNO	Numeric(3)	Left-to-right position (starting with one) of column within the table or view
COLTYPE	Char(1)	dBASE data type of column: C (character), N (numeric), D (date), F (float), or L (logical); SQL data types Smallint, Integer, Decimal, and Numeric are all recorded as type N

(continued)

Column	Type	Description
COLLEN	Numeric(3)	Length of column
COLSCALE	Numeric(2)	Number of places to the right of the decimal point
NULLS	Char(1)	Reserved for future use
COLCARD	Numeric(10)	Number of distinct values in the column (updated by CREATE INDEX and RUNSTATS for columns with COLTYPE of Numeric, Float, and Date)
UPDATES	Char(1)	Set to Y if updatable; N if not
HIGH2KEY	Char(8)	Second highest value of the column if the column is part of an index key (updated by CREATE INDEX and RUNSTATS for columns with COLTYPE of Numeric, Float, and Date)
LOW2KEY	Char(8)	Second lowest value of the column if the column is part of an index key (updated by CREATE INDEX and RUNSTATS for columns with COLTYPE of Numeric, Float, and Date)

Sysdbs Table

The Sysdbs catalog table is the *master* catalog table that keeps track of all SQL databases created with the CREATE DATABASE command. This table is different from the other catalog tables in that there is only one Sysdbs table, whereas a set of the other catalog tables is created for every new database. The Sysdbs catalog table is located in the SQL *home* directory (default directory \DBASE\SQLHOME). One row of 91 bytes is added for each new SQL database created.

Column	Type	Description
NAME	Char(8)	Name of database
CREATOR	Char(10)	User ID of person who created the database
CREATED	Date	Date the database was created
PATH	Char(64)	DOS path of database directory

Sysidxs Table

The Sysidxs table describes every SQL index and the table for which it's defined. One row of 65 bytes is added for every new index created in the current database.

Column	Type	Description
IXNAME	Char(10)	Name of index
CREATOR	Char(10)	User ID of person who created the index
TBNAME	Char(10)	Name of table on which index is defined
TBCREATOR	Char(10)	User ID of person who created table (may be different from CREATOR entry above)
UNIQUERULE	Char(1)	Set to U if index is UNIQUE; otherwise set to D (to indicate that duplicate columns are allowed)
COLCOUNT	Numeric(3)	Number of columns in index key
CLUSTERING	Char(1)	Reserved for future use
CLUSTERED	Char(1)	Statistical entry
FIRSTKCARD	Numeric(5)	Statistical entry
FULLKCARD	Numeric(5)	Statistical entry
NLEAF	Numeric(5)	Statistical entry
NLEVELS	Numeric(3)	Statistical entry

Syskeys Table

The Syskeys table describes each column defined as an index key. One row of 38 bytes exists for each column specified in the index key of an index in the current database.

Column	Type	Description
IXNAME	Char(10)	Name of index
IXCREATOR	Char(10)	User ID of person who created the index
COLNAME	Char(10)	Name of column that defines index
COLNO	Numeric(3)	Left-to-right position of column within table
COLSEQ	Numeric(3)	Relative position of index key within an index (.mdx) file
ORDERING	Char(1)	Set to A if Ascending (ASC) order; set to D if Descending (DESC) order

Syssyns Table

The Syssyns table describes synonyms defined in the current database. One row of 41 bytes is added for each new table or view synonym.

Column	Type	Description
SYNAME	Char(10)	Name of synonym
CREATOR	Char(10)	User ID of person who created synonym
TBNAME	Char(10)	Name of table or view for which synonym is defined
TBCREATOR	Char(10)	User ID of person who created the table or view for which the synonym is defined

Systabls Table

The Systabls table describes tables and views defined in the current database. One row of 74 bytes is added for each new table or view.

Column	Type	Description
TBNAME	Char(10)	Name of table or view
CREATOR	Char(10)	User ID of person who created the table or view
TBTYPE	Char(1)	Set to T if object is a base table, V if object is a view, D or K if object is a temporary table created with the SELECT... SAVE TO TEMP command (K indicates that the KEEP option was specified)
COLCOUNT	Numeric(3)	Indicates the number of columns in the table or view
CLUSTERRID	Numeric(10)	Reserved for future use
INDXCOUNT	Numeric(3)	Indicates the number of indexes defined for base tables; set to 0 for views
CREATED	Date	Date the table or view was created
UPDATED	Date	Last date that the CREATE INDEX or RUNSTATS command was run; for views, contains the same date as the CREATED column
CARD	Numeric(10)	Statistical entry
NPAGES	Numeric(10)	Statistical entry

Systimes Table

The Systimes table contains information used only in a multi-user environment to ensure that user copies of tables reflect the most recent catalog updates. One row of 27 bytes is maintained for each catalog table. (No new rows are ever added.) The DATEUP and TIMEUP columns are updated whenever a catalog table in the current database is updated.

Column	Type	Description
NAME	Char(10)	Catalog table name
DATEUP	Date	Date of last catalog table update
TIMEUP	Char(8)	Time of last catalog table update

Sysvdeps Table

The Sysvdeps table describes the tables or views on which a view is defined. One row of 36 bytes is added for each table or view on which a view is defined in the current database.

Column	Type	Description
VIEWNAME	Char(10)	Name of view
TBLNO	Numeric(5)	Left-to-right position (starting with one) of table or view in the FROM clause of the view definition
TBNAME	Char(10)	Name of table or view on which the view is defined
CREATOR	Char(10)	User ID of person who created the view

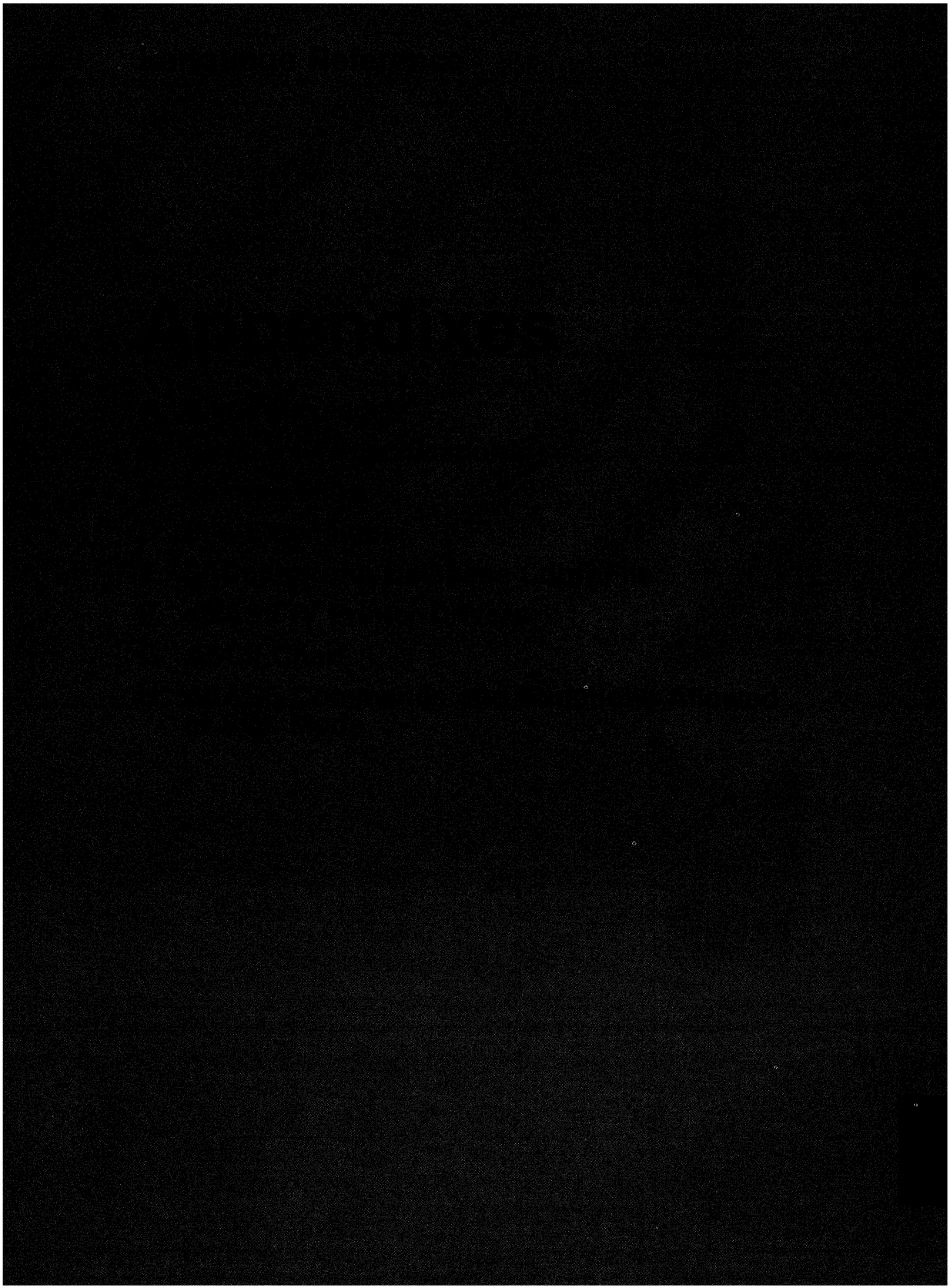
Sysviews Table

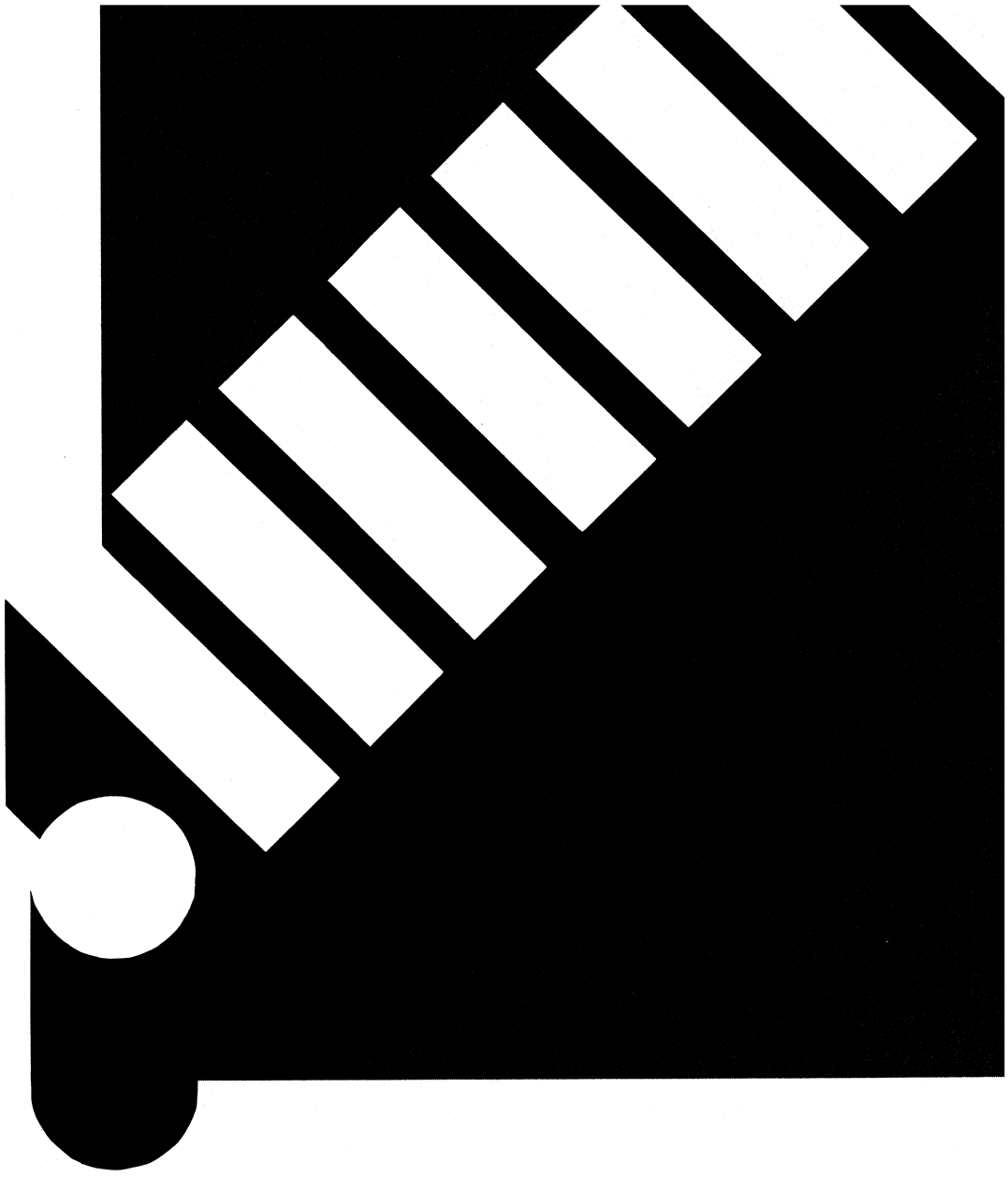
The Sysviews table describes each view. One or more rows of 225 bytes is added for each new view defined in the current database. Multiple rows are entered in Sysviews when the length of the view definition text exceeds 200 bytes. When multiple rows are entered, the Seqno column value indicates a row's relative position in the view definition.

Column	Type	Description
VIEWNAME	Char(10)	Name of view
CREATOR	Char(10)	User ID of person who created the view
SEQNO	Numeric(1)	Sequence number (starting with 1) of this row among the other rows that describe the same view

(continued)

Column	Type	Description
VCHECK	Char(1)	Set to Y if CHECK option enabled; otherwise set to N
READONLY	Char(1)	Set to Y if view is not updatable; otherwise set to N
JOIN	Char(1)	Set to Y if the view can be used in a join; otherwise set to N (if view definition contains GROUP BY clause)
SQLTEXT	Char(200)	Text of CREATE VIEW statement that defines the view

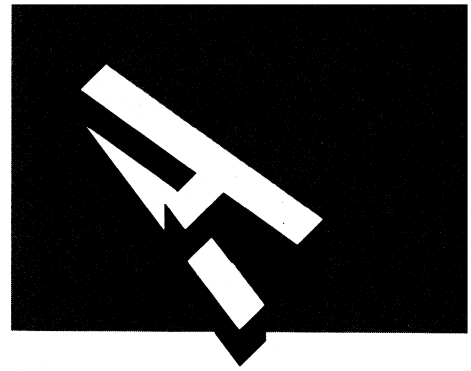




i 1 2 3 4 5 6 7 A B

C D E F G H In

Error Messages



This appendix contains all dBASE IV and SQL error messages. The messages in each section are listed in alphabetical order. The error number associated with each message is shown in square brackets at the end of the message.

dBASE Error Messages

Errors that have numbers can be trapped with the ON ERROR command, except when you resolve the error by choosing an alternative from an error box. An error-handling routine specified by ON ERROR can use the ERROR() function to return the error number, or the CERROR() function to return the number of the last compiler error. Errors that can be returned from CERROR() are marked with an asterisk (*).

Messages without error numbers originate from the Control Center and cannot be trapped.

Corrective actions are suggested whenever possible.

−: 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 −.

^ or **: Negative base, fractional exponent **[78]**
You cannot use a negative number as a parameter with arithmetic and trigonometric functions. Exponents must be integers.

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: <alias name> [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 files and 47 .mdx tags.

All database files must be closed before using PROTECT [173]
Close all databases in use before attempting to use PROTECT.

ALTERNATE <filename> 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.

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]
You provided the ATAN() function with a numeric expression that is larger or smaller than dBASE IV can compute.

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 PROCEDURE name [32]*
You are attempting to compile a procedure, but dBASE IV does not recognize the procedure name. Verify that the procedure name contains legal characters and is a valid name.

BAR position must be a positive number [167]
You must define bars of a menu or pop-up 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 under the filename of the 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 pop-up in use **[177]**
A pop-up menu active on the screen cannot be cleared with the CLEAR POP-UP or RELEASE POP-UP 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 under the filename in the file skeleton 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 in QBE.

Cannot create! SQL table exists with same name **[374]**
You are trying to create a .dbf file, and a SQL table exists with the same name. Select a different database filename.

Cannot delete SQL created TAG: <tagname> **[376]**
You may delete a SQL created index 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 link on a character field larger than 100 [315]
You tried to link QBE file skeletons on a character field whose defined length is larger than 100.

Cannot link on a memo or logical field [316]
In QBE, you entered an example variable in a memo or logical field.

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 a 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 the bottom (left edge, right edge, or top) of the layout
You tried to move the cursor off the design screen.

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]
Production .mdx files may contain tags created in SQL mode and dBASE mode. This message indicates that you tried to create an index tag in dBASE mode with the same name as an existing SQL index tag.

Cannot perform an update if errors exist [395]
You chose to **Perform the update** in QBE, but errors exist in the query definition.

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 DEFINE a MENU with the same name as an active menu. Issue a DEACTIVATE MENU command first, then use DEFINE MENU.

Cannot re-define popup in use **[175]**
You are attempting to DEFINE a POPUP with the same name as an active popup. 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 or create a production .mdx file for SQL system catalog files. SQL system catalog files are read-only, therefore dBASE IV cannot update the .dbf file header to indicate that a production .mdx file exists. You can, however, create non-production .mdx files.

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 cannot be called recursively **[103]**
A LIST or DISPLAY command cannot recursively call another LIST or DISPLAY. For example, you cannot use LIST with a UDF, if the UDF calls another LIST or DISPLAY. A BROWSE cannot call another BROWSE (or APPEND, CHANGE, EDIT, or INSERT) in the same work area. See SET DBTRAP in Chapter 3 for further information about recursive commands.

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]***
Some commands may not be allowed in a UDF, depending on the setting of SET DBTRAP. See SET DBTRAP in Chapter 3 for further information.

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 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.

Coordinates are off the screen **[30]**
You are using row and column numbers that are off the screen when defining or moving a window. In 25-line mode, the screen limits are 0,0 to 24,79.

Copying production .MDX requires one free work area **[487]**
If you use the PRODUCTION option of the COPY command, one work area must be available. Close a database file in one work area, then re-issue the COPY command.

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.

dBASE IV editor will not save memo fields larger than 64K **[137]**
You are trying to save a memo field larger than the dBASE IV editor's 64K limit.

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]**
or
Disk <drive> is full. Delete old files from disk.
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 or 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. Use 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 or 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 WHEN 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.

Environment not correct for rollback **[193]**
A database file referenced in the transaction log file cannot be found. Locate the database file or files involved in the transaction and issue a ROLLBACK command with the name of each .dbf file. After you ROLLBACK changes to all .dbf files, verify that the transaction log file has been deleted. If not, QUIT (or use the RUN command) to delete the log file from the operating system level.

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 *Getting Started* 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. See RESET in Chapter 2.

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 and 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 *Getting Started.*

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 *Getting Started.*

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 *Getting Started.*

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. Examples:

*****Execution error on SUBSTR(): Start point out of range**

*****Execution error on STR(): Out of range**

^--- Expected ON or OFF [73]
The SET command you are changing can only accept ON or OFF as its parameter setting.

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 neglected to provide a field name during CREATE or MODIFY STRUCTURE.

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: <field name> [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. If you are USEing a database file, the file may be open in another work area.

File cannot be modified because DESIGN mode is set to OFF [314]
This message usually occurs in programs that prevent you from accessing dBASE IV design mode. SET DESIGN OFF prevents you from creating or modifying files from the Control Center or by using dBASE IV commands, and also disables use of the **Shift-F2** key combination in APPEND, BROWSE, and EDIT. SET DESIGN ON if you must create or modify a file. See SET DESIGN.

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 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]
The file you attempted to open is not a valid .mdx file.

File does not exist: <filename> [1]
The specified file cannot be found. Verify the filename, and check the current path for the file's existence.

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 half of an example variable pair
In QBE, you entered FIRST before both halves of an example variable pair, such as FIRST x in one file skeleton and FIRST x in another.

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 UPDATE from a database only when that database is not in the selected work area.

Function not found: <function name> **[368]**
The user-defined function cannot find the associated FUNCTION procedure in the currently open program and procedure files.

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 or 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 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 10 characters or less.

Illegal field width
You have entered an illegal field length for the current field in database design.

Illegal Key Expression **[386]**
The key expression cannot be used to create an index. For example, the use of UDFs (user-defined functions) in a key expression depends on the setting of SET DBTRAP.

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. This limitation refers to the number of characters in the key expression, rather than the actual length of the resulting key.

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]
The result of the key expression you INDEXed on exceeds 100 characters. This limitation refers to the actual length of the key after the expression is evaluated.

Index TAG already exists: <tag name> [205]
The index tag you are trying to use already exists; select another tag.

Insufficient memory [43]*
An operation failed because you have reached the limits of available RAM. Add more memory, or remove memory resident utilities and restart dBASE IV.

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 IV 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 IV 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 or 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 last 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 [3
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 RunTime command, omitted from .DBO **[348]**
A command that cannot be used in RunTime was encountered, but it will not be compiled into the .dbo file.

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: <tag name> **[284]**
This message means that the name you have specified for the tag name is not in the .mdx file.

^--- Keyword not found **[86]**
A keyword you used in the Config.db file is not a valid dBASE keyword. See *Getting Started* for a list of valid keywords.

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: <filename> **[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 nested too deep: <macro calling sequence> **[361]**
You attempted 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 memo field size exceeded
The maximum size of a memo field is 64K.

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 48th index to the .mdx file by setting more than 47 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 print width of 255 characters exceeded
This occurs when label width times number of labels exceeds 255 characters.

Maximum record length exceeded in WK1 file **[392]**
Your Lotus worksheet exceeds the maximum record length of 4,001 bytes.

Maximum sort number is 9 **[309]**
You entered a sort number greater than 9, such as **ASC10**, in a QBE file skeleton.

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 tags in an .mdx file 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 chips. If you are not familiar with computer hardware, contact a service technician to diagnose and replace the bad memory chips.

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 the minimum size for that field type.

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 half of example pair **[363]**
In a QBE file skeleton, you entered an example variable without entering the required matching example variable in another file skeleton. For example, if you enter "WITH x" in a REPLACE query, you must define x in another file skeleton. Verify that both halves of the example pair are spelled correctly.

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 is not a valid dBASE field name. This also occurs if you try to add an unnamed calculated field to the view.

NDX index limit reached **[204]**
You can have ten .ndx or .mdx 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 numberic expression.

Not a dBASE database [15]
You are trying to USE a database file that is not in the correct dBASE IV format. This message may also mean that you tried to IMPORT a dBASE II database file without first changing its extension to .db2.

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 Found **[82]**
This message occurs in Browse or Edit in a full-screen operation when a field cannot be found.

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 move the cursor from below the file skeleton, 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 below the filename in a calculated field skeleton.

Operation not allowed with a user-defined function **[150]**
Certain operations not allowed in user-defined functions are described in Chapter 1, "Essentials," and under SET DBTRAP in Chapter 3.

Operator not allowed in calculated fields [313]
You entered an AVG, COUNT, GROUP BY, MAX, MIN, or SUM operator in a QBE calculated fields skeleton.

Operator not allowed in target file [333]
You entered an AVG, COUNT, GROUP BY, MAX, MIN, or SUM operator in a QBE file skeleton that is the target file for an update query.

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: <tag name> [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.

Original memo cannot be larger than 64K [411]
You are trying to append new text to a memo that is already larger than 64K.

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.

^--- Out of range [75]
The parameter you are using with this function exceeds the allowed range. Check Chapter 4, "Functions," for the function range.

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 executable 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 prompt, or with an @ command, ? command, or other display commands) 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 is too large (> 64K) [258]*
A compiled procedure is limited to exactly 65,520 bytes. Your procedure is too large. Divide the procedure into smaller functional groups.

Procedure not found: <procedure name> [252]
A procedure with this name cannot be found in the currently open program or procedure files.

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. dBASE IV asks you if you want to continue.

Production MDX file not found: <filename> [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 length of the fields in a unique query is greater than 100 characters, or if the command line generated by the query engine is greater than 1,024 characters.

If the sum of the field lengths of all fields selected in the view skeleton is greater than 100, you must remove one or more fields from the view skeleton.

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.

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 record in the related database file is locked by another user.

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 create exceeds the limit of 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 certain ON commands.

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 **[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 must match index **[330]**
In a QBE file skeleton, you entered a sort operator into a complex index column, but the sort direction does not match the index direction. For example, if the index is descending, you cannot enter ASC1.

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 a SQL statement.

SQL System Tables may not be USEd [218]
You have attempted to USE a 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 less than zero or greater than the length of the input 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 the 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]
In addition to the production .mdx file, you can open a maximum of 10 .ndx and .mdx files in a work area.

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 length of the fields in a unique query is greater than 100 characters. As the sum of all the field lengths of selected fields must not be greater than 100, you must select only those fields needed to make the query unique.

TRAP can not be turned OFF while Debugger is active [390]
You must SET TRAP ON or OFF before you call the debugger program.

^--- 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.

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.

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.

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 command verb **[16]***
You are using a command that is not part of dBASE IV syntax.

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.

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 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 names must be 10 characters or less [312]
You entered a variable name longer than 10 characters in a QBE file skeleton.

VARIABLE not found: <variable name> [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 ** Data will probably be lost. Confirm? (Y/N)** [70]
This message occurs after an error message notifies you that the disk is full. You have the choice of deleting other files from disk or abandoning the operation.

WARNING: File definition has changed [320]
The structure of one of the files has been modified since you last modified the query.

WARNING: If field altered, CONVERT information will be lost
If you delete or modify the _dbaselock field on a CONVERTed database file, CHANGE() and LKSYS() cannot be used.

**** 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: 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.

**** WARNING ** Uncompleted transaction found** [190]
You are attempting to use the BEGIN TRANSACTION command before first issuing an END TRANSACTION command.

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. If DISPLAY = EGA43, the coordinates must be between 0,0 and 38,79.

WINDOW has not been defined [214]
You are attempting to activate a window that has not been defined. Define the window first.

Window is locked: <window name> [184]
An intervening command, as in a UDF or ON routine, cannot RELEASE or CLEAR the WINDOW that was active when the interruption occurred.

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 is locked: <alias name> [409]
A command is processing the current work area, and an intervening command (perhaps in a UDF or ON routine) is seeking to close the database file in the work area. The database file must remain open for the first command to resume processing.

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 PARAMETERS keyword in the program, or the WITH keyword in the DO command, is followed by an incorrect number of parameters.

Zoom window command editor, CTRL-END exits to dot prompt
This is an information message from the zoom window command editor.

SQL Error Messages

Below is the list of error messages for SQL commands in dBASE IV. Error numbers for messages that can be trapped with the ERROR() function are listed to the right of the error message in square brackets.

NOTE
If errors occur during the execution of a SQL program, follow the advice pertaining to the displayed error message. If the error still occurs, you can do the following:

1. *Run DBCHECK to check that database file and index structures match entries in corresponding SQL catalog tables.*
2. *Run RUNSTATS to update statistical information in the SQL catalog tables.*
3. *Recompile the programs that caused an error.*

*** allowed for COUNT function only** **[1082]**
An asterisk has been used as the argument to an aggregate function other than COUNT(). The argument for functions AVG(), MAX(), MIN(), and SUM() must be a column name.

ADD clause expected in ALTER TABLE **[1036]**
ADD clause missing, misspelled, or misplaced in ALTER TABLE command.

Aggregate function not allowed in WHERE clause **[1103]**
The SQL aggregate functions AVG(), COUNT(), MAX(), MIN(), and SUM() are not allowed in a WHERE clause (except as part of the SELECT clause in a subselect). You may want to redefine the operation using a subselect. For example, *SELECT * FROM Staff WHERE Salary > AVG(Salary)* can be replaced by *SELECT * FROM Staff WHERE Salary > (SELECT AVG(Salary) FROM Staff)*.

Alias name already exists: <alias name> **[1083]**
You have already applied this alias name to another table or view in the same clause. Use a different alias name.

All SELECT columns must be inside an aggregate function **[1164]**
When HAVING is used without GROUP BY, all SELECT columns must either be constants, or be inside a SQL aggregate function. Place all column names inside an aggregate function, remove them from the SELECT clause, or add a GROUP BY clause.

All SELECT items must be GROUP BY columns or aggregate functions **[1100]**
A SELECT clause includes both aggregate functions (AVG(), COUNT(), MAX(), MIN(), and SUM()) and column names which are not included in aggregate functions. All columns not included in aggregate functions must be included in a GROUP BY clause. Add a GROUP BY clause to the command. If the GROUP BY clause is already part of the command, add the missing SELECT clause column names to it or add an aggregate function to the column name.

ALTER privilege not granted
Grantor does not possess the ALTER privilege, or did not receive it WITH GRANT OPTION.

ALTER privilege not revoked
The user revoking this privilege never granted it to the user from whom it is to be revoked, or the privilege has been revoked already.

Ambiguous column name: <column name> **[1077]**
A column with the same name appears in two tables, both referenced in the current statement. Prefix ambiguous columns with their table or alias name and a period, as in <table/alias name>.<column name>.

An illegal table is referenced in a subselect FROM clause: <table name> **[1115]**
The table upon which the INSERT, UPDATE, or DELETE command is carried out cannot also be referenced in the FROM clause of a subselect.

Argument too long in CREATE INDEX/GROUP BY/ORDER BY/ UNION/SELECT DISTINCT **[1185]**
An index expression of more than 100 characters has caused an error on a CREATE INDEX command, or on a SELECT command including clauses (such as GROUP BY or ORDER BY) that use indexes. Specify a shorter index key for a CREATE INDEX command. If this error occurred on a SELECT command, it is an internal error. You can remove the GROUP BY or ORDER BY clause, or reduce the number of columns in DISTINCT or UNION.

Badly formed subquery **[1172]**
A predicate in an outer query is placed in a nested query. Rewrite the query without a subquery (that is, use the outer predicate to limit the rows that are retrieved).

BY clause is not supported and will be ignored
This is a warning, not an error. dBASE IV SQL does not support a BY clause in the GRANT and REVOKE statements. No action needed.

Cannot ALTER views: <view name> **[1151]**
Only base tables and their synonyms may be used in an ALTER TABLE statement. You can DROP a view, then redefine it to include additional columns added to the base table.

Cannot CREATE INDEX/GROUP BY/ORDER BY/SELECT DISTINCT/ UNION on a LOGICAL col. **[1182]**
Index cannot be built on a column of LOGICAL data type and all these operations involve internal SQL indexes. Remove the LOGICAL type column.

Cannot DROP open database: <database name> **[1141]**
You tried to DROP the active database. First use STOP DATABASE and then use the DROP DATABASE command.

Cannot GRANT or REVOKE a privilege to yourself **[1201]**
A GRANT or REVOKE command specifies your own user ID in the TO or FROM clause. You may only GRANT and REVOKE privileges of other users. Check the user ID list in your statement and make sure it does not contain your own user ID. Make sure you logged in with your own user ID.

Cannot LOAD or UNLOAD DATA for views **[1143]**
LOAD and UNLOAD DATA commands must be used with SQL base tables or their synonyms. You may use a SELECT statement with a SAVE TO TEMP clause on the view and then UNLOAD from this temporary table.

Cannot mix ASC and DESC options in index key **[1183]**
If the keyword DESC is specified, it must be specified for all keys in the index.

Cannot read the header of the following file: <filename>
An error occurred during execution of a DBCHECK, DBDEFINE, or RUNSTATS because the header of a .dbf file could not be read. If the error occurred on DBDEFINE, remove the file with the bad file header from the database directory before re-executing DBDEFINE. If the error occurred on DBCHECK or RUNSTATS, you must DROP the table in SQL before you can successfully re-execute the command.

Can't create subdirectory for new database **[1155]**
The path or drive in your statement does not exist, or you tried to create a database with the same path as SQLHOME. Verify the path and database name.

Catalog table Sysdbs does not exist **[1229]**
The Sysdbs table must be in the SQLHOME directory. Make sure that you have not deleted the Sysdbs.dbf file and that the SQLHOME path in Config.db is correct. If SQLHOME has been changed from its Install setting, make sure the complete set of SQL system tables, including Sysdbs, has been transferred to the new SQLHOME directory.

Catalog tables are read-only: <table name> **[1161]**
Catalog table updates are only allowed on a password-protected system by the SQLDBA superuser ID.

Catalog table(s) locked by another user: <table name> **[1140]**
Locks on the SQL catalog tables prevent completion of an operation. Wait and retry operation. If this message appears often, SET REPROCESS to a higher number of retries.

Character string too long **[1145]**
A character string exceeds the dBASE maximum length for character data. Shorten the string.

CHECK OPTION cannot be used with current view **[1107]**
The WITH CHECK OPTION clause cannot be used with a view that cannot be updated. See the CREATE VIEW command entry in Chapter 6 for rules on updating views.

Column data type doesn't match for table, column: <table>
An error occurred on a DBCHECK or RUNSTATS command because the data type indicated in the Coltype column of the Syscols catalog table doesn't match the data type of the actual column. First, copy the .dbf file and its .mdx or .dbt to another directory and DROP the SQL table. Then, copy all files back to the database directory and use DBDEFINE <filename> to redefine the table and indexes.

Column is not updatable: <column name> **[1109]**
An INSERT or UPDATE command specifies a derived column or a column in a non-updatable view. See the CREATE VIEW command entry in Chapter 6 for rules on updating columns.

Column name already exists: <column name> **[1081]**
A command that names new columns has repeated the same name twice, or has attempted to use a name that already exists for some column in the table. Use a different column name.

Column name missing in AVG, MAX, MIN, or SUM function **[1130]**
The SQL aggregate functions AVG(), MAX(), MIN(), and SUM() require a column name as their argument.

Column name or number expected in ORDER BY **[1020]**
Only column names, or integers can appear in the ORDER BY list. After the keyword ORDER BY, either give the columns by which you want the result table ordered, or substitute integers indicating the columns according to their place in the SELECT clause.

Column not found in catalog table Syscols for table column: <table and column names>
An error occurred on DBCHECK or RUNSTATS command because a field in the .dbf file was not found as a column in the Syscols catalog table. First, copy the .dbf file and its .mdx file to another directory and DROP the SQL table. Then, copy the .dbf and .mdx files back to the database directory and use DBDEFINE <filename> to redefine the table and indexes.

Column/field names must be specified in SAVE TO TEMP clause **[1149]**
The SELECT returns columns derived from functions or constants, so SAVE TO TEMP column names must be specified. Add a list of column names enclosed in parentheses after the filename in the SAVE TO TEMP clause.

Column's position doesn't match for table, column: <table>
An error occurred on DBCHECK or RUNSTATS command because the information in the Syscols catalog table about a column's position in a table doesn't match the actual position of the column in the table. First, copy the .dbf file and its .mdx file to another directory and DROP the SQL table. Then, copy the .dbf file and its .mdx file back to the database directory and use DBDEFINE <filename> to redefine the table and indexes.

Comma or right parenthesis expected **[1052]**
A list has been specified incorrectly. Check for commas between the items and for a right parenthesis at the end of the list.

Command cannot be executed within a transaction **[1167]**
SQL data definition commands ALTER, CREATE, DBCHECK, DBDEFINE, DROP, GRANT, REVOKE, and RUNSTATS are not allowed in transactions. Run these commands outside of transactions.

Comparison operator or keyword expected **[1134]**
A comparison operator or keyword must follow the first column name or constant in a WHERE clause. The comparison operators are: =, >, <, >=, <=, <>, !=, !<, or !>. The comparison keywords are NOT, LIKE, IN, and BETWEEN.

Correlated subquery not allowed in HAVING clause **[1160]**
A subselect in a HAVING clause cannot reference the same table as the outer query. Restructure the query into two or more simpler queries, using tables created by the SAVE TO TEMP command, if necessary.

Cursor already open: <cursor name> **[1125]**
An OPEN command in a .prs program specifies a cursor that has already been OPENed. Precede the OPEN command with a CLOSE command.

Cursor declaration does not include the FOR UPDATE OF clause **[1123]**
An error has occurred during execution of an UPDATE WHERE CURRENT OF statement or DELETE WHERE CURRENT OF statement, because the DECLARE CURSOR command that defined the cursor used for the UPDATE did not include a FOR UPDATE OF clause. Include the FOR UPDATE OF clause in the DECLARE CURSOR statement of the cursor used for the UPDATE. Note that the ORDER BY clause cannot be used when the FOR UPDATE OF clause is included.

Cursor name previously declared: <cursor name> **[1133]**
A DECLARE CURSOR statement contains a cursor name that has already been used in another DECLARE CURSOR statement in the same .prg program. Choose a new name for the cursor.

Cursor not declared: <cursor name> **[1121]**
No DECLARE CURSOR statement defines this cursor. Include a DECLARE CURSOR statement in your .prs program. Make sure the DECLARE CURSOR statement precedes the first statement that references the cursor.

Cursor not open: <cursor name> **[1124]**
A FETCH or CLOSE CURSOR command in a .prs program cannot be executed because the specified cursor has not been OPENed. Precede the FETCH or CLOSE command with an OPEN command.

Cursor not updatable: <cursor name> **[1120]**
The DECLARE CURSOR statement that defined the cursor included a SELECT DISTINCT, a UNION, aggregate functions, or included more than one table, or a non-updatable view. Use that cursor only for displaying records. DECLARE another cursor for use in DELETE/UPDATE WHERE CURRENT OF and INSERT statements.

Data type keyword expected **[1033]**
In a CREATE TABLE or ALTER TABLE command, the data type of a column must be specified after the column name. The SQL data types are: SMALLINT, INTEGER, DECIMAL, NUMERIC, FLOAT, CHAR, LOGICAL, and DATE. If data type is already specified, check for misspelling, or for misplacement of a keyword or parenthesis.

Data type mismatch of corresponding columns in UNION operation **[1093]**
The corresponding columns of SELECT statements joined by the UNION keyword are not of matching data types. Check the data types of corresponding columns: SMALLINT, INTEGER, DECIMAL, NUMERIC, and FLOAT types are considered matching for UNION operations; however, they must specify the same length and number of decimal places. Other data types must match exactly.

Database does not exist **[2002]**
A database may have been DROPped since the program was compiled.

Database in use by another — cannot drop **[1248]**
You cannot issue DROP DATABASE if another user on the network has activated that database.

Database name already exists: <database name> **[1136]**
The database name used in a CREATE DATABASE command already exists as a SQL database. Use a different name for the new database.

DBCHECK and RUNSTATS must be used with base tables **[1225]**
A DBCHECK or RUNSTATS command specifies a view name or a non-.dbf filename instead of a base table name. To display the list of non-view tables in the active database, use the following query: *SELECT Tbname FROM Systabls WHERE Tbtype = "T".*

DBDEFINE completed; errors/warnings found **[2030]**
At least one file was not converted to a SQL table because of inconsistencies in the SQL catalog tables, or problems in the file's structure, or because the same filenames already exist in the system catalogs. Informational messages preceding the error indicate which tables have been successfully DBDEFINEd and which have not.

DELETE privilege not granted
Grantor does not possess the DELETE privilege, or did not receive it WITH GRANT OPTION.

DELETE privilege not revoked
The user revoking this privilege never granted it to the user from whom it is to be revoked, or the privilege has been revoked already.

Delimiter must be one character long or keyword BLANK **[1048]**
After the keywords DELIMITED WITH in a LOAD or UNLOAD utility, you must specify BLANK or a single character as delimiter.

Different table name is specified in cursor declaration: <cursor name> **[1126]**
The table specified in the DECLARE CURSOR statement is not the same as the table specified by the UPDATE/DELETE WHERE CURRENT OF statement using that cursor. Check table and cursor names to make sure the same table is referenced in both the DECLARE CURSOR and UPDATE/DELETE WHERE CURRENT OF statements.

DISTINCT must be followed by a column name not of type LOGICAL **[1097]**
When the keyword DISTINCT is used with the SQL aggregate functions, it must be followed by a non-LOGICAL type column name.

Duplicate user ID **[1202]**
The GRANT or REVOKE command user ID list contains duplicate user IDs. Remove duplicate user IDs.

Duplicated value in unique index — aborted **[2000]**
UPDATE or INSERT was not performed because it would have duplicated a value in a unique index key. Change the new value so that it is distinct from all current values. You may also DROP the index, but consider carefully before doing so, since unique indexes are used to insure database integrity.

Entry already exists in catalog table Syscols for the following table, column: <table and column names>
DBDEFINE has attempted to define a table for which an entry already exists in the Syscols catalog table. This error indicates severe inconsistencies in the database directory catalog tables. CREATE a new database directory, copy all non-catalog .dbf and .mdx files to a new database, and use DBDEFINE; or modify the column name with the dBASE IV MODIFY STRUCTURE command before using DBDEFINE.

Entry already exists in catalog table Sysidxs for the following table, index: <table and index names>
DBDEFINE has attempted to define an index for which an entry already exists in the Sysidxs catalog table. Error indicates severe inconsistencies in the database directory catalog tables. CREATE a new database directory, copy all non-catalog .dbf files and related .mdx files to a new database, and use DBDEFINE; or create the tag with a new tag name in dBASE mode before using DBDEFINE.

Entry already exists in catalog table Syskeys for the following table, index, column: <table, index, column names>
DBDEFINE has attempted to define an index key for which an entry already exists in the Syskeys catalog table. Error indicates severe inconsistencies in the database directory catalog tables. CREATE a new database directory, copy all non-catalog .dbf and related .mdx files to the new database, and use DBDEFINE; or create the tag on a different expression in dBASE mode before using DBDEFINE.

Equal sign expected **[1022]**
The equal sign is missing or misplaced after the keyword SET <column name> in an UPDATE statement.

Expression not allowed in GROUP BY/ORDER BY **[1186]**
Only column names are allowed in GROUP BY lists. Only column names or integers are allowed in ORDER BY lists. Remove all expressions from the clause.

File cannot be open in dBASE mode for this operation **[2011]**
While using a data definition command, such as CREATE or DROP, the corresponding table or system tables are open in dBASE mode. Close the tables in dBASE mode with CLOSE or USE before proceeding.

File encryption error **[1282]**
File encryption error. Check the encrypted file. You can SET ENCRYPTION OFF and use a SQL LOAD or dBASE COPY TO command to create an unencrypted copy of a file.

File has invalid SQL encryption **[1228]**
DBCHECK or RUNSTATS has encountered a file that has invalid SQL encryption. Try to copy an unencrypted version of the file using the SQL UNLOAD command. If this is successful, erase the encrypted version, use DBDEFINE <filename> and then re-execute the DBCHECK or RUNSTATS command. If UNLOAD is not successful, the table must be DROPped in SQL before DBCHECK or RUNSTATS will execute successfully.

File is encrypted **[1226]**
DBDEFINE cannot be used with an encrypted file. If the file is dBASE-encrypted, SET SQL OFF. Use the dBASE IV COPY command to create an unencrypted copy of the file. Erase the encrypted version before re-executing DBDEFINE. If the file is SQL-encrypted, use the SQL UNLOAD command to create an unencrypted copy of the file. Erase the encrypted version before re-executing DBDEFINE.

File is not legal dBASE/SQL: <database filename> **[1230]**
An error has occurred during DBCHECK, DBDEFINE, or RUNSTATS execution because file header information indicates it is not a dBASE or SQL file. Make sure that you have not inadvertently copied a non-dBASE or SQL file with a .dbf extension. Remove the file causing the error from the SQL database, before re-executing the DBCHECK, DBDEFINE, or RUNSTATS utility command.

File is not SQL encrypted **[1227]**
DBCHECK or RUNSTATS has encountered a file that is encrypted, but not under the SQL encryption key. File must be decrypted by the user or removed before re-executing the command.

Filename is same as existing synonym **[1217]**
Error occurs on DBDEFINE when the .dbf file specified in the command has the same name as an existing SQL synonym. Rename the .dbf file before re-executing DBDEFINE <filename>, or drop that synonym.

File not found in the current database **[1231]**
An error occurred during execution of DBCHECK, DBDEFINE, or RUNSTATS command because the file was not found in the currently active database directory. Check that you have not misspelled a filename, and that the named .dbf file is in the active database directory.

File open error: <filename> **[1280]**
File open error.

File read error: <filename> **[1277]**
File read error.

File seek error: <filename> **[1276]**
File seek error.

File write error: <filename> **[1278]**
File write error.

First argument of LIKE clause must be a CHAR column **[1090]**
The column name preceding the keyword LIKE in a WHERE clause does not identify a CHARACTER type column. Do not use LIKE with a non-CHARACTER type column.

Float value out of range **[1146]**
A float value greater than 10^{+38} or smaller than 10^{-38} cannot be entered.

GRANT OPTION ignored for UPDATE with column list specified
This is a warning, not an error. The grantee will not be able to GRANT the UPDATE privilege to others because the UPDATE privilege was GRANTed only for certain columns. No action needed. If the UPDATE privilege is GRANTed without a column list, then the WITH GRANT OPTION will become effective.

GRANT OPTION ignored when GRANT is TO PUBLIC
This is a warning, not an error. The GRANT OPTION is ignored because the privileges are being GRANTed to PUBLIC and so no further GRANTs will be necessary. No action needed.

GROUP BY clause needed **[1129]**
The SELECT clause includes both SQL aggregate functions (AVG, COUNT, MAX, MIN, or SUM) and column names. All column names in the SELECT clause that are not part of the aggregate functions must be included in a GROUP BY clause.

GROUP BY column(s) not specified in the SELECT clause **[1094]**
A column has been specified in the GROUP BY clause that is not included in the SELECT clause. Include the GROUP BY columns in the SELECT clause. All non-GROUP BY columns in the SELECT clause must be columns derived from aggregate functions.

GROUP BY, HAVING, ORDER BY, UNION, or FOR UPDATE OF clause not allowed with INTO **[1128]**
A SELECT statement in a .prs program includes both the INTO clause and GROUP BY, HAVING, ORDER BY, UNION, or FOR UPDATE OF clauses. A SELECT statement including the INTO clause should only return a single row and cannot include any of these clauses.

HAVING clause must include aggregate functions **[1116]**
The HAVING clause specifies a search condition for grouped data, and usually follows a GROUP BY clause. You must specify at least one of the aggregate functions AVG(), COUNT(), MAX(), MIN(), or SUM() in the HAVING clause.

Host variable count in INTO clause is not equal to number of SELECT items **[1119]**
A SELECT statement in a .prs program contains an INTO clause with a number of variables that does not match the number of columns in the SELECT clause.

In UNION, ORDER BY column(s) must be specified by integers **[1096]**
You may use an ORDER BY clause to order the result table of two or more SELECT statements joined by a UNION, but you must use integers instead of column names since column names may be different in each SELECT clause. Replace column names with integers (for example, to ORDER BY the first column, use 1, the second column, use 2, and so on).

Incompatible data types in comparison **[1132]**
The data types of columns, constants, or expressions in a comparison do not match. Check the data types of columns used. Check for missing quotes on string constants.

Incompatible data types in expression **[1131]**
Columns or constants with incompatible data types have been used in an expression. CHAR, LOGICAL, and DATE types cannot be mixed with the data types that hold numeric values, or with each other. Check the data types of columns used in the expression to make sure they are compatible with other columns, constants, or functions used in it.

Incorrect data type for arguments in dBASE function **[1112]**
Refers to a dBASE function used in a SQL statement. Check Chapter 4 for the correct data types in the dBASE function.

Incorrect number of arguments in dBASE function **[1113]**
Refers to a dBASE function used in a SQL statement. For example, this error might occur when the CTOD() function is used with an invalid value for the date, as in CTOD("99/99/99"). Check Chapter 4 for correct syntax and usage of the dBASE function.

Incorrect number of INSERT items **[1078]**
The number of values from the VALUES clause or the subselect doesn't match the number of columns specified in the column list (or in the table if no column list was given). Check that the column list includes the exact columns for which data is to be INSERTed. Check the VALUES list or subselect SELECT clause to make sure the correct number of values are provided.

Index name already exists **[1102]**
The index name used in a CREATE INDEX command already exists as a SQL index. Use a different index name for the new index, because each index name must be unique in a SQL database.

INDEX privilege not granted
Grantor does not possess the INDEX privilege, or did not receive it WITH GRANT OPTION.

INDEX privilege not revoked
The user revoking this privilege never granted it to the user from whom it is to be revoked, or the privilege has already been revoked.

INSERT privilege not granted
Grantor does not possess the INSERT privilege, or did not receive it WITH GRANT OPTION.

INSERT privilege not revoked
The user revoking this privilege never granted it to the user from whom it is to be revoked, or the privilege has been revoked already.

Insufficient memory [1275]
Insufficient memory available for allocation. Simplify queries by breaking long statements into several shorter ones.

Insufficient privilege [1204]
You do not have the privilege to perform the requested operation on the table or view specified. Have the creator of the table or view GRANT you privileges on it. Anyone who received the privileges WITH GRANT OPTION may also GRANT them to you.

Insufficient privilege to CREATE VIEW [1205]
To CREATE a view, you must have SELECT privileges for every table on which the view is based. Remove the table for which you lack SELECT privileges from the view definition. Or, have someone GRANT you SELECT privileges on the table.

Integer expected [1032]
A non-integer value was encountered. Replace with an integer value.

Internal SQL error #1 [1261]
Undefined nodename.

Internal SQL error #2 [1262]
String table overflow. Simplify your query.

Internal SQL error #3 [1263]
Illegal SQL command.

Internal SQL error #4 [1264]
Internal relation table overflows. Simplify your query.

Internal SQL error #5 [1265]
Error in parse tree. Simplify your query.

Internal SQL error #6 [1266]
A unique index has been used, and more than 10 unique indexes use the same column as part of their unique key. Drop some unique indexes.

Internal SQL error #7 [1267]
Internal file open error.

Internal SQL error #8 [1268]
Internal relation table overflow. Simplify your query.

Internal SQL error #9 [1269]
Too many columns created in temporary relations. Simplify your query.

Internal SQL error #10 [1270]
Optimizer join class error. Simplify your query.

Internal SQL error #11 [1271]
Temporary Systabl overflows in optimizer. Simplify your query.

Internal SQL error #12 [1272]
Temporary SYSCOLUMNS overflows in optimizer. Simplify your query.

Internal SQL error #13 [1273]
Error in temporary system catalogs in optimizer. Simplify your query.

Internal SQL error #14 [1274]
Invalid pointer to free allocated memory.

Internal SQL error #15 [1275]
Insufficient memory available for allocation. Simplify queries by breaking a long command into several shorter ones.

Internal SQL error #16 [1276]
File seek error.

Internal SQL error #17 [1277]
File read error.

Internal SQL error #18 [1278]
File write error.

Internal SQL error #19 [1279]
Internal hash table for system catalogs overflow. Simplify your query.

Internal SQL error #21 [1281]
Systimes error.

Internal SQL error #22 [1282]
File encryption error. Check file encryption. You can SET ENCRYPTION OFF and use a SQL UNLOAD or dBASE COPY TO command to create an unencrypted copy of a file.

Internal SQL error #24 [1284]
Query array overflow.

Internal SQL error #29 [1245]
The emitted dBASE source line is longer than 1024 bytes. Check for an * in a SELECT clause or check for a complex WHERE predicate, in particular IN, ANY, or ALL.

Internal SQL utility error #1 [1218]
Error in dBASE routine that returns information to DBCHECK, DBDEFINE, or RUNSTATS.

Internal SQL utility error #2 [1219]
Bad file pointer encountered during execution of DBCHECK, DBDEFINE, or RUNSTATS.

Internal SQL utility error #3 [1220]
Bad structure pointer encountered during execution of DBCHECK, DBDEFINE, or RUNSTATS.

Internal SQL utility error #4 [1221]
Bad index pointer encountered during execution of DBCHECK, DBDEFINE, or RUNSTATS.

Internal SQL utility error #5 [1222]
Bad column pointer encountered during execution of DBCHECK, DBDEFINE, or RUNSTATS.

INTO clause not allowed in cursor declaration [1127]
The SELECT clause in a DECLARE CURSOR statement cannot include the INTO clause. Remove the INTO clause.

INTO is not allowed in a view definition [1106]
The INTO keyword has been included in a CREATE VIEW command. Remove INTO from your view definition.

Invalid argument for aggregate function **[1099]**
You have used an asterisk (*) or a column of a disallowed data type in the AVG(), MAX(), MIN(), or SUM() function. Columns of data type Logical cannot be used in SQL aggregate functions. The argument of SUM() and AVG() functions must be a column or expression that yields a numeric value.

Invalid arithmetic expression **[1135]**
An arithmetic expression was expected. Check expression for syntax errors.

Invalid character **[1002]**
Only letters, digits, and underscores are allowed in object and column names. Numbers may contain only digits and decimal points. Object names must start with a letter. Delete any disallowed characters.

Invalid character length **[1145]**
The length of a character string either equals zero, or exceeds the maximum length of 255 for dBASE character strings.

Invalid column number in ORDER BY clause **[1084]**
A column number is not an integer, or is greater than the number of columns returned by the SELECT clause. Make sure the column number corresponds to the column's placement in the SELECT clause (for example, the first column is designated by the number 1, the second by 2, and so on).

Invalid constant **[1011]**
A SQL statement is expecting a constant but no value is specified. Check for missing parentheses or commas in values lists.

Invalid COUNT argument **[1037]**
Count argument must be (*) or (DISTINCT/ALL <column name>).

Invalid decimal length **[1159]**
The CREATE TABLE command incorrectly specifies the length of a Numeric or Decimal type column. The width and scale of Numeric and Decimal type columns is specified as (x,y). Both column types may be specified as (1,0). If y is not 0, then y may not be greater than x−2. For both types, y may not exceed 18.

Invalid file type specified **[1050]**
In the LOAD and UNLOAD statements file type may be specified as SDF, DIF, WKS, SYLK, FW2, RPD, DBASEII, or DELIMITED. Check the file type. If the file LOADed from or UNLOADed to is a .dbf file, the type need not be specified.

Invalid filename **[1049]**
An invalid filename appears in the LOAD or UNLOAD statement. Check that the filename (and path) are correctly specified. When the file type is .dbf, the extension need not be specified.

Invalid INSERT item **[1152]**
Items in the value list following the keyword VALUES cannot be column names or compound expressions involving arithmetic operators. Check the value list to make sure it contains only the following items: constants, dBASE functions, memory variables, or the USER keyword.

Invalid logical predicate **[1168]**
This message occurs when you use comparison operators or keywords other than the unary equal (=) or not equal (!=, <>, or #) operators in a predicate evaluating a logical type column, memory variable, or array element.

Invalid password: <user ID> **[1283]**
Invalid password in file header.

Invalid SQL statement **[1059]**
In the stand-alone version, the first word in a command is not recognized as a SQL keyword. Check the syntax of the command.

Invalid string operator **[1150]**
Operators other than plus and minus cannot be used with strings. Remove disallowed operator. Also check that you have not mistakenly identified something as a string by inadvertently starting a name with a quotation mark.

Invalid SYSTIME.MEM **[1285]**
The SYSTIME.MEM time stamp in the active database directory is invalid. Make sure you have not accidentally overwritten the Systime.mem file. Try copying it from a backup of the database.

Invalid unary operator **[1157]**
An operator has been inappropriately applied to a non-numeric value. Check for typographical errors or operators in front of CHAR, LOGICAL or DATE columns.

Keys not unique, index not created **[2006]**
A CREATE INDEX command specified the UNIQUE option, but the columns on which the index was to be built contain duplicate values. Add a column to the index key, or DELETE rows containing duplicate index key values before re-executing the CREATE INDEX with UNIQUE option.

Keyword AND expected **[1018]**
The keyword AND was expected after first argument in BETWEEN clause. Check that the keyword AND is not missing, misspelled, or misplaced.

Keyword AS expected **[1028]**
The keyword AS is missing, misspelled, or misplaced in CREATE VIEW command. AS should follow the view name (and optional column list) and precede the subselect.

Keyword ASC or DESC, comma, or right parenthesis expected **[1057]**
The CREATE INDEX command is incomplete. Add the ASC or DESC option if desired after the column names in the column list (use commas to separate items in the list) and be sure a right parenthesis ends the column list.

Keyword ASC or DESC, comma, or semicolon expected **[1058]**
An ORDER BY clause is incomplete. ASCending, or DESCending order may be specified after the column names in the ORDER BY clause. ASC is the default. The statement must be terminated with a semicolon (;).

Keyword BY expected **[1016]**
The keyword ORDER must be followed by the keyword BY. Check that BY is not missing, misspelled, or misplaced.

Keyword CHECK expected **[1038]**
The keyword CHECK is missing, misspelled, or misplaced in the WITH CHECK OPTION clause of a CREATE VIEW command.

Keyword CURSOR expected **[1041]**
The keyword CURSOR is missing, misplaced, or misspelled in the DECLARE CURSOR statement in a .prs program.

Keyword DATA expected **[1040]**
Missing keyword DATA in LOAD or UNLOAD statement. Check that the word DATA is directly after the word LOAD or UNLOAD and that it is spelled correctly.

Keyword DATABASE expected **[1051]**
The keyword DATABASE is missing, misplaced, or misspelled in a START, STOP, or SHOW DATABASE command.

Keyword DATABASE, TABLE, INDEX, SYNONYM, or VIEW expected **[1027]**
A keyword is missing or misspelled after a CREATE keyword. Use DATABASE, TABLE, INDEX, SYNONYM, or VIEW depending on the kind of object to be CREATED.

Keyword DATABASE, TABLE, INDEX, SYNONYM, or VIEW expected **[1031]**
A keyword is missing or misspelled after DROP keyword. Use DATABASE, TABLE, INDEX, SYNONYM, or VIEW depending on the kind of object to be DROPped.

Keyword FOR expected **[1030]**
The keyword FOR is missing, misplaced, or misspelled in the CREATE SYNONYM or DECLARE CURSOR command.

Keyword FROM expected **[1015]**
The keyword FROM is missing, misplaced, or misspelled in a LOAD, UNLOAD, SELECT, REVOKE, or DELETE command.

Keyword GRANT expected **[1054]**
The keyword GRANT is missing, misplaced, or misspelled in the WITH GRANT OPTION clause of a GRANT statement.

Keyword INDEX expected **[1017]**
The keywords CREATE UNIQUE must be followed by the word INDEX. Check that the keyword INDEX is not missing, misspelled, or misplaced.

Keyword INTO expected **[1023]**
The keyword INTO is missing, misplaced, or misspelled in a FETCH, INSERT, or LOAD DATA statement.

Keyword not allowed in interactive mode **[1118]**
A keyword used during an interactive SQL session can only be used in .prs programs. All keywords related to the use of SQL cursors are used only in programs. Replace the command with an interactive SQL command.

Keyword OF expected **[1043]**
The keyword OF is missing, misplaced, or misspelled in a FOR UPDATE OF or WHERE CURRENT OF clause.

Keyword ON expected **[1026]**
The keyword ON is missing, misspelled, or misplaced in a CREATE INDEX, GRANT, or REVOKE command.

Keyword OPTION expected **[1039]**
The keyword OPTION is missing, misspelled, or misplaced in the WITH GRANT OPTION clause of a GRANT command, or in the WITH CHECK OPTION clause of a CREATE VIEW statement.

Keyword SELECT expected **[1029]**
The keyword SELECT is missing, misplaced, or misspelled after the keyword UNION, or in a CREATE VIEW command after the keyword AS. The missing SELECT keyword is the first word of the required subselect used in these statements.

Keyword SELECT missing in DECLARE CURSOR statement **[1046]**
The keyword SELECT is missing, misplaced, or misspelled in a DECLARE CURSOR statement. A SELECT statement must follow the keyword FOR.

Keyword SET expected **[1021]**
The keyword SET is missing, misplaced, or misspelled in UPDATE statement.

Keyword TABLE expected **[1035]**
The keyword TABLE is missing, misspelled, or misplaced in an ALTER TABLE, LOAD, or UNLOAD command.

Keyword TEMP expected **[1053]**
The keyword TEMP is missing, misplaced, or misspelled in the SAVE TO TEMP clause of a SELECT statement.

Keyword TO expected **[1025]**
The keyword TO is missing, misspelled, or misplaced in a GRANT or UNLOAD command or the SAVE TO TEMP clause of a SELECT command.

Keyword UPDATE expected **[1042]**
The keyword UPDATE is missing, misspelled, or misplaced in the FOR UPDATE OF clause in the DECLARE CURSOR statement.

Keyword VALUES or SELECT expected **[1024]**
The INSERT statement must include either a VALUES clause or a subselect. Check that one of the keywords is included and that it is not misspelled or misplaced.

Keyword WITH expected **[1047]**
The keyword WITH is missing, misplaced, or misspelled in the LOAD or UNLOAD utility where delimiters are specified. When data is LOADed to or UNLOADed from dBASE files, the TYPE and WITH clauses need not be specified.

Keywords BETWEEN, LIKE, or IN expected **[1019]**
A WHERE clause is incorrectly specified. Check for correct syntax of the desired form of the WHERE clause.

Left parenthesis missing **[1013]**
A left parenthesis is missing before the beginning of a list. Check the syntax of the command for required parentheses.

Length of column doesn't match for table, column: <table and column name>
An error occurred on a DBCHECK or RUNSTATS command because the column length indicated in the Collen column of the Syscols catalog table does not match the actual length of the column. First, copy the .dbf file and its associated .mdx file to another directory and DROP the SQL table. Then, copy the .dbf and .mdx files back to the database directory and use DBDEFINE <filename> to redefine the table and indexes.

Memory variable and dBASE function not allowed in SELECT with UNION **[1163]**
All SELECT columns must be named columns or constants when UNION is used. Remove memory variables, dBASE functions, and array references from SELECT clauses.

Memory variable or column name undefined or memory variable of invalid type **[2007]**
A name has been found that is not a column name, and has not been defined as a memory variable, or the memory variable data type does not match that expected in the SQL statement. Check that a column name or memory variable name has not been misspelled. Also, check that columns belong to the referenced table; make sure that values have been STOREd to memory variables and that those values are compatible with the data types expected in the SQL statement.

Missing end quotes for string **[1003]**
Strings must be enclosed in quotes. Add missing quote (″). Verify that you have not inadvertently used a quote that has been interpreted as a marker to begin a string. Also verify that single and double quote marks are not mixed.

Name already exists: <object name> **[1080]**
The name used in a CREATE TABLE, CREATE SYNONYM, or CREATE VIEW command already names an existing SQL table, synonym, or view. Use a different name for the new table, synonym, or view.

Name, constant, or expression expected **[1010]**
A scalar value was expected. Check that a column name, constant, or expression has not been omitted from the statement. Look for incorrectly placed commas; commas should not be used to separate clauses.

Name expected **[1012]**
An object name is missing, misplaced, or misspelled. Check that database, table, view, synonym, or column names are correctly specified. Also, check for incorrectly placed commas; commas should not be used to separate clauses.

Name longer than 8 characters not allowed **[1156]**
The name for a database, table, synonym, or view exceeds eight characters. Use a name of eight characters or less (beginning with a letter and containing only letters, digits, and underscores).

Name longer than 10 characters **[1001]**
SQL column names, index names, and memory variables referenced in SQL statements may be up to 10 characters long. (Database, table, view, synonym, and index names may be up to eight characters long.)

Name longer than 20 characters **[1001]**
All SQL names must contain 20 or fewer characters. (Database and table names must contain 8 or fewer.) Names must begin with a letter and contain only letters, digits, and underscores. Check for column names that are too long.

Nested function not allowed **[1098]**
An aggregate function contains another nested aggregate function. Remove the nested aggregate function.

No alias defined for self-join: <table name> **[1142]**
When a self-join form of the WHERE clause is used, an alias must be defined for each table in the FROM clause. The table name cannot be referenced twice, nor may a synonym name be used in place of an alias. Use a FROM clause of the form FROM <table name> <alias name>. Then prefix column names with aliases in the SELECT and WHERE clauses as necessary.

No ALTER or INDEX privileges for views **[1203]**
These privileges cannot be GRANTed or REVOKEd because views cannot be ALTERed or INDEXed. You cannot GRANT or REVOKE ALL PRIVILEGES on views because ALL includes the ALTER and INDEX privileges. A privilege list for views may contain only the following privileges: DELETE, INSERT, SELECT, or UPDATE.

No current row available for UPDATE or DELETE: <table or view name> **[1154]**
This error occurs only in .prs programs when UPDATEs and DELETEs are being performed under SQL cursor control. Make sure a FETCH has been executed before the UPDATE or DELETE is attempted.

No database open **[1139]**
The command requested can only be executed when a SQL database is open. If you have entered SQL with all databases closed, or if you used the STOP DATABASE command, the current command cannot be executed. Use a START DATABASE command before retrieving information.

No .dbf file in the current database **[1224]**
No .dbf files (other than the SQL catalog tables) were found during execution of a DBDEFINE command. Make sure that the .dbf files that you want to define as SQL tables have been copied into the current database directory before executing the DBDEFINE command.

No entry found in catalog table Sysidxs for the following table, index: <table and index names>
An error occurred on a DBCHECK or RUNSTATS command because an index file was found with a name that does not exist in the Sysidxs catalog table. First, copy the .dbf file and its associated .mdx file to another directory and DROP the SQL table. Then, copy the .dbf and .mdx files back to the database directory and use DBDEFINE <filename> to redefine the table and indexes.

No entry found in catalog table Syskeys for following table, index: <table, index, column name>
An error occurred on a DBCHECK or RUNSTATS command because an index key has been found with an index name that does not exist in the Syskeys catalog table. First copy the .dbf file and its associated .mdx file to another directory and DROP the SQL table. Then, copy the .dbf and .mdx files back to the database directory and use DBDEFINE <filename> to redefine the table and indexes.

No record selected
A query returned no rows.

Non-numeric array subscript **[1162]**
All array subscripts must evaluate to integers. If columns or functions are used, check that the data type of the expression is numeric.

Number of columns doesn't match in table: <table name>
An error occurred on a DBCHECK or RUNSTATS command because the actual number of columns in a table does not match the number of columns indicated in the Col-count column of the Systabls SQL catalog table. First, copy the .dbf file and its associated .mdx file to another directory and DROP the SQL table. Then, copy the .dbf and .mdx files back to the database directory and use DBDEFINE <filename> to redefine the table and indexes.

Number of columns in index key doesn't match for table, index: <table and index names>
An error occurred on a DBCHECK or RUNSTATS command because the Sysidxs catalog table indicates a different number of columns in the index key from the actual number of columns in the index. First, copy the .dbf file and its associated .mdx file to another directory and DROP the SQL table. Then, copy the .dbf and .mdx files back to the database directory and use DBDEFINE <filename> to redefine the table and indexes.

Number of columns must be the same in UNION operation [1092]
The SELECT statements joined by the UNION keyword do not generate the same number of columns. Change the SELECT clauses of the SELECT statements so that they return the same number of columns. Columns must also be of matching data type and length.

Number of decimal places doesn't match for table, column: <table and column names>
An error occurred on a DBCHECK or RUNSTATS command because the scale indicated in the Colscale column of the Syscols catalog table doesn't match the scale of the actual column. First, copy the .dbf file and its associated .mdx file to another directory and DROP the SQL table. Then, copy the .dbf and .mdx files back to the database directory and use DBDEFINE <filename> to redefine the table and indexes.

Number of indexes doesn't match for table: <table name>
An error occurred on a DBCHECK or RUNSTATS command because the number of indexes indicated by the Systabls catalog table does not match the actual number of indexes. First, copy the .dbf and associated .mdx files in question to another directory, and DROP the table from the current database. Then, copy the .dbf and .mdx files back into the current database and use DBDEFINE to redefine them. Then re-execute the DBCHECK or RUNSTATS command.

Number of SAVE TO TEMP columns does not match number of SELECT columns [1148]
The SAVE TO TEMP column list must contain the same number of columns as the SELECT statement results being saved. If all the columns in the result table are named columns (not derived from functions, constants, and so on), then no column list need be specified in the SAVE TO TEMP clause.

Number of view columns does not match number of SELECT columns [1104]
In a CREATE VIEW command, the number of columns specified in the view column list is not the same as the number of columns generated in the SELECT clause. If all the columns in the view are named columns (not the result of aggregate functions, expressions, and so on), then the column list may be omitted and the view columns will inherit SELECT column names.

Numeric value too large [1144]
A value exceeds dBASE limits on numeric values.

Numeric value too small [1153]
A value is smaller than the smallest allowable dBASE numeric value.

Only one column may be SELECTed in a subquery [1101]
A subquery has returned multiple values. Use the predicate IN, ANY, or ALL in the WHERE clause to process the values, or change the SELECT clause of the subquery to return only one column.

Only one DISTINCT allowed in any SELECT clause **[1114]**
The keyword DISTINCT has appeared more than once in a SELECT clause. Remove the DISTINCT option.

ORDER BY clause not allowed in CREATE VIEW **[1056]**
An ORDER BY clause was included as part of the AS SELECT in a CREATE VIEW statement. The ORDER BY clause may only be used in a full SELECT, with UNION, or in the DECLARE CURSOR statement. Remove the ORDER BY clause. You may use an ORDER BY clause when you SELECT from the view.

ORDER BY column(s) not specified in the SELECT clause **[1095]**
A column has been specified in the ORDER BY clause that is not included in the SELECT clause. Include the ORDER BY columns in the SELECT clause.

Ordering of index column doesn't match for table, index, column: <table, index, and column names>
An error occurred on a DBCHECK or RUNSTATS command because the Syskeys catalog table indicates an ordering of ASC or DESC that is not the actual ordering of the column in the index. First, copy the .dbf file and its associated .mdx file to another directory and DROP the SQL table. Then, copy the .dbf and .mdx files back to the database directory and use DBDEFINE <filename> to redefine the table and indexes.

Path too long **[1044]**
An operating system path name may not exceed 64 characters.

Position of column in index key doesn't match for table, index, column: <table, index, column names>
An error occurred on a DBCHECK or RUNSTATS command because the position of a column in an index key as indicated in the Syskeys catalog table doesn't match the column's actual position in the index. First, copy the .dbf file and its associated .mdx file to another directory and DROP the SQL table. Then, copy the .dbf and .mdx files back to the database directory and use DBDEFINE <filename> to redefine the table and indexes.

PROTECT cannot be used on SQL tables
SQL tables must be protected with GRANT/REVOKE. You cannot encrypt a SQL table with PROTECT.

Right bracket missing **[1060]**
A left bracket appears without a matching right bracket. Check array references to make sure brackets match. Square brackets are not used elsewhere in SQL.

Right parenthesis missing **[1014]**
A left parenthesis is present without a matching right parenthesis. Check for missing right parenthesis in subselects, aggregate functions, and so on.

Rows violates view definition — INSERT/UPDATE row rejected **[2005]**
A row cannot be or UPDATEd or INSERTed in a view because the view was created WITH CHECK OPTION and the inserted or updated row violates the view's definition. You can read the view's definition by SELECTing the Sqltext column from the Sysviews catalog table. You can also INSERT into or UPDATE the underlying base table, but such rows will not appear in the view.

RUNSTATS completed without catalog updates error/warning found **[2009]**
RUNSTATS was not successful because of inconsistencies in the database. Note that no updates are made, even for tables for which no error or warning appeared. Check the error messages that preceded this message, follow suggested remedies, and run RUNSTATS again.

SAVE TO TEMP clause not allowed **[1147]**
SAVE TO TEMP clause not allowed as part of a DECLARE CURSOR statement, or when an INTO clause is used in a SELECT statement.

Second argument of LIKE clause must be a character string **[1091]**
LIKE must be followed by a character string, the USER keyword, or a character-type memory variable. Functions and column names are not allowed.

SELECT cannot include both FOR UPDATE OF and ORDER BY clauses **[1045]**
A SELECT or DECLARE CURSOR statement cannot include both the FOR UPDATE OF and ORDER BY clauses. If a cursor defined in a DECLARE CURSOR statement is to be used in an UPDATE WHERE CURRENT OF statement, then the FOR UPDATE OF clause must be chosen. Otherwise, drop it and keep the ORDER BY clause.

SELECT column names must be in aggregate functions **[1163]**
When HAVING is used without GROUP BY, all column names in the SELECT clause must be operated on by aggregate functions. Add a GROUP BY clause that includes all SELECT clause column names, or include all SELECT clause column names in aggregate functions.

SELECT privilege not granted
Grantor does not possess the SELECT privilege or did not receive it WITH GRANT OPTION.

SELECT privilege not revoked
The user revoking this privilege never granted it to the user from whom it is to be revoked, or the privilege has been revoked already.

SQL index cannot contain more than 10 columns **[1173]**
You cannot create a SQL index with more than 10 column references.

SQL syntax error **[1034]**
A name, keyword, operator, comma, or semicolon is missing or misspelled; or a UNION clause is used in the view definition. Check for a missing semicolon after the SQL statement, and for the correct syntax of the command. Remove the UNION clause from your view definition.

SQLHOME is not set in CONFIG.DB **[1250]**
If you SET SQL ON (or include SQL=ON in the Config.db file), the directory containing the SQL files must be declared in the Config.db file. Modify Config.db to include SQLHOME =<path name>, then restart dBASE IV.

Subquery did not return any value **[2011]**
The subquery in a WHERE or HAVING clause did not return any values. (Only the WHERE EXISTS predicate may include a subquery that doesn't return any values.) Modify the search condition subquery. You can test the query separately to verify whether it returns the correct values.

Subquery did not return exactly one value **[2003]**
A subquery following an arithmetic comparison operator in a WHERE clause must return only one value. An error occurred because multiple values were returned. Either precede the subquery with the keyword ANY or ALL, or rewrite the WHERE clause using the keyword IN instead of an arithmetic comparison operator.

System table entry missing for table **[2004]**
The Systabls catalog table does not contain an entry for this table; it has been DROPped, or does not exist.

Systimes error **[1281]**
The Systimes.dbf file (used to indicate updates to tables in a multi-user environment) has been corrupted. Have each user currently accessing the associated database CLOSE the affected database. Then, transfer a new copy of the Systimes.dbf file from the SQLHOME directory to the affected SQL database directory.

Table already exists **[1216]**
The filename following the DBDEFINE keyword already exists as a SQL table, so no new entries can be made in the SQL catalog tables for this file. If you need to redefine this table (because of DBCHECK errors, or for any other reason), you must copy the .dbf file and its associated .mdx file to another directory and DROP it as a SQL table. Copy the .dbf and .mdx files back into the database directory, then use DBDEFINE <filename>.

Table(s) not DBDEFINEd:
Lists files which were not converted to SQL tables. See the **DBDEFINE completed; errors/warnings found** error message.

Table not found in the SQL catalog tables **[1223]**
The table name specified in the DBCHECK or RUNSTATS command does not exist in the SQL catalog tables. Verify the correct table name with a SELECT from Systabls catalog table.

Table not included in current statement: <table name> **[1075]**
A column prefixed with the table name from which it comes is referenced, but its table is not included in the current FROM clause. Add the missing table name to the FROM clause.

The following file was not found in the current database: <filename>
On DBCHECK or RUNSTATS there is an entry for a table in the SQL system catalogs, but there is no .dbf file with this name in the database directory. The file may have been erased outside of SQL (either from dBASE IV or from the operating system).

The following table already exists: <table name>
The filename specified in a DBDEFINE command already exists in the SQL catalog tables.

The following table name is the same as the existing synonym: <table name>
This error occurs when running DBDEFINE if one of the .dbf files to be defined has the same name as an existing SQL synonym. Rename the .dbf file before re-executing DBDEFINE, or drop that synonym.

The following table was not found in catalog table SYSTABLS: <filename>
An error occurred on a DBCHECK or RUNSTATS command because a .dbf was found with a name that does not appear in the Systabls catalog table. Either remove the .dbf file from the database directory or use the DBDEFINE <filename> command to define the .dbf file as a SQL table before re-executing the DBCHECK or RUNSTATS command.

This type of correlated subquery not allowed **[1160]**
The SQL statement contains correlated unequal joins, correlated joins on logical columns, or correlated joins with a DISTINCT option. Remove the unequal joins, joins on logical columns, or the DISTINCT option. Use a temporary table created by SAVE TO TEMP to retrieve data before doing the joins.

Too many columns in a table **[1166]**
This message appears if a CREATE TABLE or ALTER TABLE command attempts to create a table with more than 255 columns.

Too many indexes for a table **[1170]**
You cannot create more than 47 indexes for a single table. Drop indexes you no longer need before creating new indexes.

Too many unique indexes **[1266]**
A unique index has been used and more than ten unique indexes all use the same column as part of their unique key. You need to drop some of the unique indexes.

Too many values specified in INSERT **[1085]**
A column list was specified that contained more columns than exist in the table. Check the column list for duplicate column names and remove extra columns

Too many work areas open **[2008]**
Either more than 10 tables are referenced in one SQL statement, or you have left some work areas open before issuing the SET SQL ON command and the total number of open work areas is greater than 10. Reference fewer tables in the FROM clause, reduce the complexity of the SQL query, or return to dBASE mode and close work areas.

Unable to open SQL system catalog: <SQL catalog filename> **[1247]**
The system catalog is missing or corrupted. Verify that the catalog file is available.

Unable to open the SYSDBS file in the SQL Home Directory **[1249]**
The SYSDBS file is missing or corrupted.

Undefined column name: <column name> **[1076]**
The column name that caused this error is not a column in any of the tables referenced in the command. Either you have referenced the wrong table, forgotten to include a table in the FROM clause, misspelled the column name, or used a column name that doesn't exist.

Undefined database name: <database name> **[1137]**
There is no entry in the Sysdbs catalog table for this database. Check that you have not given an incorrect database name, or use a CREATE DATABASE statement to create the database.

Undefined index name: <index name> **[1088]**
The index named in a DROP INDEX command does not exist as a SQL index. (The index name is not entered in the SQL catalog table Sysidxs.) Do a SELECT operation on the Sysidxs catalog table to verify the correct name of the index.

Undefined privilege in GRANT or REVOKE statement **[1055]**
You may GRANT and REVOKE the following privileges: SELECT, INSERT, DELETE, UPDATE, INDEX, ALTER. Check for misspellings or keywords such as CREATE that cannot be GRANTed.

Undefined symbol **[1004]**
An unrecognized or inappropriate symbol has been accidentally included in the statement. Check for and delete any symbols such as $ or &, or arithmetic comparison or operator symbols accidentally included in names or keywords.

Undefined synonym name: <synonym name> **[1089]**
The synonym named in a DROP SYNONYM command does not exist as a SQL synonym. (The synonym name is not entered in the Syssyns catalog table.) Check that you have not given a table or view name by mistake. Use a SELECT operation on the Syssyns catalog table to get a listing of all existing synonym names.

Undefined table name: <table name> **[1086]**
The table, synonym, or view requested is not entered in the relevant SQL catalog table. Either the name is misspelled, or a CREATE (or DBDEFINE) command was never used. Check that you have not put a column name or other item where a table name was expected. Do a SELECT from the appropriate catalog table (Systabls, Syssyns, or Sysviews) to list the correct names.

Undefined view name: <view name> **[1087]**
The view named in a DROP VIEW command is not a SQL view. (The view name is not entered in the Sysviews catalog table.) Do a SELECT from the Sysviews catalog table to verify existing view names.

UPDATE column list is ignored in REVOKE statement
This is a *warning*, not an error. REVOKEing the UPDATE privilege for a table applies to all columns of the table. After REVOKEing the UPDATE privilege, GRANT it again for specific columns.

UPDATE column(s) not defined in cursor declaration: <column name> **[1122]**
The UPDATE WHERE CURRENT OF statement in a .prs program updates columns that were not included in the column list of the FOR UPDATE OF clause of the associated DECLARE CURSOR statement. Add all columns to be updated to the column list in the FOR UPDATE OF clause of the DECLARE CURSOR statement.

UPDATE may only be REVOKEd from the whole table, not by column
This is a *warning*, not an error. REVOKE of an UPDATE privilege applies to all columns in a table. After REVOKEing the UPDATE privilege, GRANT it again specifying the columns to which it applies.

UPDATE privilege not granted
Grantor does not possess the UPDATE privilege, or did not receive it WITH GRANT OPTION.

UPDATE privilege not revoked
The user revoking this privilege never granted it to the user from whom it is to be revoked, or the privilege has been revoked already.

Value exceeds column length **[1158]**
A value is longer than the column into which it is to be entered. Query the Syscols catalog table to determine the length of the column into which you are entering values.

Value must match column data type **[1079]**
An UPDATE or INSERT value doesn't match the data type of the column into which it is to be placed. Check the data type of the column. Be sure dates are correctly entered in the form CTOD(" / / "), or { / / }. Check for missing beginning quotes on character strings.

View column names must be specified **[1105]**
The SELECT clause of a CREATE VIEW statement includes columns derived from aggregate functions, dBASE IV functions, constants, memory variables, or from two different tables. View column names cannot simply be inherited from the SELECT clause. Add a list of column names enclosed in parentheses after the CREATE VIEW keywords.

View defined with GROUP BY cannot be used in a join **[1110]**
A view referenced in the FROM clause of a statement joining views (and tables) has a GROUP BY clause as part of its definition. This view cannot be used in a join.

View defined with GROUP BY cannot be used in a query with a GROUP BY clause **[1111]**
A SELECT statement including a GROUP BY clause references a view that has a GROUP BY clause as part of its definition. Remove the GROUP BY clause in the current SELECT, or do not use the named view in this query.

View is not updatable: <view name> **[1108]**
An update operation was attempted on a non-updatable view. Views with definitions referencing more than one table, or including GROUP BY, SELECT DISTINCT, or aggregate functions are not updatable.

Views cannot be INDEXed: <view name> **[1117]**
A CREATE INDEX command included a view name instead of a table or synonym name. Views cannot be INDEXed.

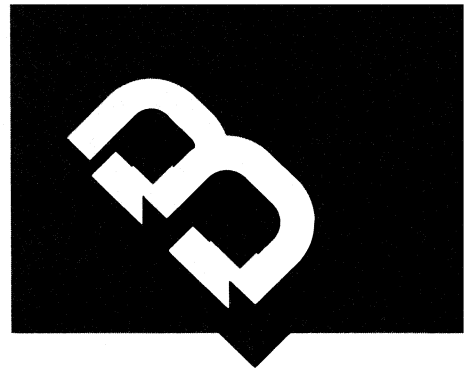
Warning — Cursor <cursor name> not closed
Safety message. An open cursor was not closed by the user before the end of the program or procedure. Upon exiting a program or procedure to the dot or SQL prompt, SQL automatically closes all cursors opened during the program or procedure.

Warning — DROP DATABASE will drop all SQL tables
Safety warning box that appears when you enter a DROP DATABASE command. If you continue, all SQL tables, both catalog tables and data tables, will be deleted. The database's entry in the Sysdbs catalog table is also deleted. Choose the **Cancel** option to abandon the operation, or **Proceed** to continue DROPping the database.

Warning — No WHERE clause specified in DELETE
Safety prompt that appears if a DELETE command contains no WHERE clause. If executed, this DELETE will delete all rows in the table. Choose the **Cancel** option to abandon the operation, or **Proceed** to continue deleting all rows.

Warning — No WHERE clause specified in UPDATE statement
Safety prompt that appears if an UPDATE command contains no WHERE clause. If executed, the command will update all rows in the table. Choose the **Cancel** option to abandon the operation, or **Proceed** to continue updating all rows.

dBASE IV Specifications



This appendix outlines the specifications for dBASE IV files and operations.

Database File

Number of records: 1,000,000,000
Number of bytes: 2,000,000,000
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
Memo fields: 64K

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: 11 (10 .ndx, 1 .mdx)
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 file)	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 available 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

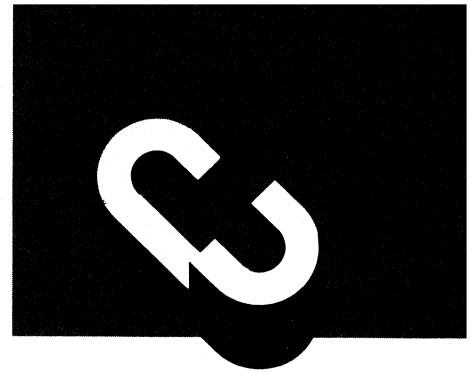
Number of objects on work surface: depends on complexity of objects and available memory
Size of full-screen editing frame: 4K

Miscellaneous Capacities

Command line length: 255 bytes at dot prompt, 1,024 bytes in edit window
Maximum nesting level of control commands: determined by available memory
Procedures per program or 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: 16
Number of windows: 20

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   : 10/17/88
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    : 10/17/88
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   : 10/17/88
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-111-6050 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, which produces a horizontal bar menu and four pop-up menus:

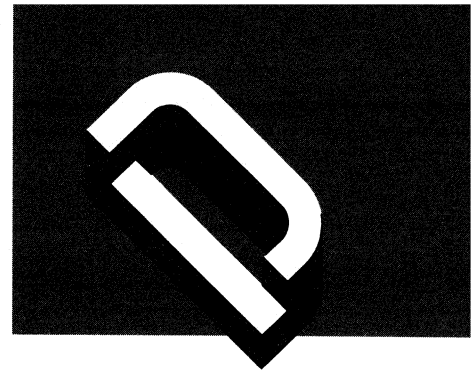
```
* Menus.prg
SET TALK OFF
CLEAR
Medit = .F.
DO Def_mens
ON PAD View OF Main ACTIVATE POPUP View_pop
ON PAD Goto OF Main ACTIVATE POPUP Goto_pop
ON PAD Print OF Main ACTIVATE POPUP Prin_pop
ON SELECTION PAD Exit OF Main ACTIVATE POPUP Exit_pop
ON SELECTION POPUP Exit_pop DO Exit_pro
ON SELECTION POPUP 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
```

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 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	Program (.prg) or procedure 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 and COPY
.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 (.frg) file
.fw2	Framework spreadsheet or database file; used for import and 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
.out	Default file extension for output of template language commands
.pop	Pop-up menu design object file; Applications Generator only
.pr2	Printer driver file
.prf	Print form file
.prg	dBASE program or procedure file

(*continued*)

Table D-1 dBASE IV file extensions (*continued*)

Extension	File Content
.prs	dBASE SQL command or procedure file
.prt	Printer output file
.qbe	QBE query file
.qbo	Compiled QBE query (.qbe) file
.qry	Query file
.rpd	RapidFile file; used for import and export, and APPEND FROM and COPY
.sc3	Renamed old dBASE III screen (.scr) file
.scr	Screen file
.slk	Multiplan spreadsheet file; used with APPEND FROM and COPY
.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 and COPY

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

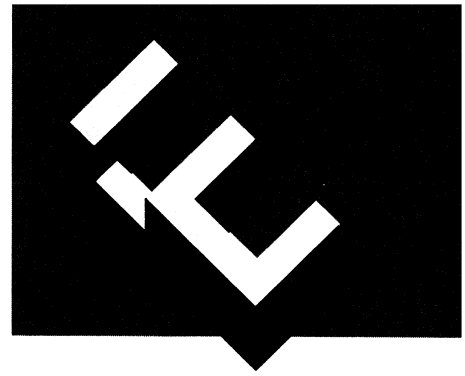
File Type	dBASE IV Extension	Compiled Object File	Backward Compatibility	Rename from dBASE III PLUS
Catalog	.cat		yes	
Database	.dbf		yes *	
Memo	.dbt		no **	
Form, design	.scr		no	.sc3
Form, code generation	.fmt	.fmo	no ***	
Report, design	.frm		no	.fr3
Report, code generation	.frg	.fro	no	
Label, design	.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	

* If your dBASE IV database files contain type F numeric fields, use COPY TO <filename> TYPE DBMEMO3. This command converts the type F fields to type N, so the new file will contain only type N numbers. dBASE III PLUS can use type N numbers, but not type F numbers. 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 allows only 19 digits in a numeric field.)

** You can use COPY TO <filename> TYPE DBMEMO3 to convert the memo file back to dBASE III PLUS format.

*** 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, bit 3 indicates the presence of a dBASE IV memo file, bits 4–6 indicate the presence of a SQL table, bit 7 indicates the presence of any memo file (either dBASE III PLUS or dBASE IV)
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
21–30	10 bytes	Reserved
31	1 byte	Production .mdx field flag; 01H if field has an index tag in the production .mdx file, 00H if not

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

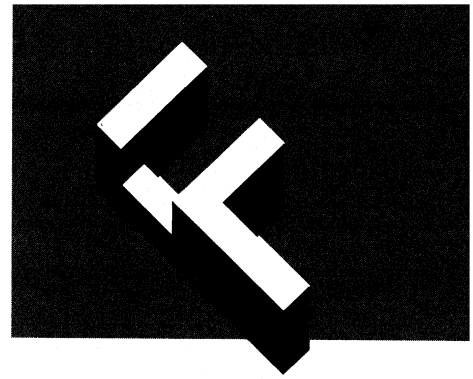
A memo (.dbt) file consists of blocks numbered sequentially (0, 1, 2, and so on). SET BLOCKSIZE determines the size of each block. The first block in the memo file, block 0, is the memo file header.

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 *Getting Started* 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. In this table, *Box Characters* means the driver prints the ASCII characters between decimal 169 and 218. If box characters are not supported, boxes are printed with dashes (–), ASCII 45, and piping characters (¦), ASCII 124.

Table F-1 dBASE IV printer drivers

Manufacturer	Model	Driver Filename
AMT	AMT 250	AMT.pr2
Apple	LaserWriter[1]	POSTSCRI.pr3 POSTSCRI.dld[2]
Anadex	6000 9625B (Silent Scribe)	ANA6000.pr2 ANA9625B.pr2
ASCII[3]		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 1550S, SC 715F 815 (P351 Mode) 8510A, B, BPI 8510S, SC 3500 model 20 ProWriter Jr.	CI1550A.pr2 CI1550S.pr2 CI715__2.pr2 CI815.pr2 CI8510A.pr2 CI8510S.pr2 CI3520.pr2 FX80__1.pr2
Datasouth	180 220	DS180.pr2 DS220.prR
DataProducts	P-80, P-132 SPG-8050, SPG-8070	DP__P80.pr2 DP__8050.pr2
Diablo	(Xerox) 630 API	DIAB630A.pr2
Epson America, Inc.	FX-85, FX-185 FX-86E GQ-3500 JX-80 JX-80 with 8177 LX-80, LX-90 MX-80, MX-100 MX-80/MX-100 with Graftrax PLUS RX-80, FX-80, FX-80 PLUS RX-80/FX-80 with 8177 RX-100, FX-100, FX-100 PLUS RX-100/FX-100 with 8177 LQ-800 LQ-1500, SQ-1500 DX-10 DX-20, DX-35	FX85__1.pr2 FX86E__1.pr2 GQ35__150.pr2 JX80__2.pr2 EPSN8177.pr2 LX80.pr2 Generic.pr2 MX80G.pr2 FX80__1.pr2 EPSN8177.pr2 FX80__1.pr2 EPSN8177.pr2 LQ800__2.pr2 LQ1500__1.pr2 DX10.pr2 DIAB630A.pr2
Facit	4512	FAC4512.pr2
Fujitsu	DotMax 9f DotMax 9i DotMax 24i DotMAx 24c	FX80__1.pr2 IBMGP.pr2 FUJ24I.pr2 FUJ24C.pr2
Generic Driver [4]	Any printer not listed	GENERIC.pr2

[1] Or fully compatible printer using Adobe PostScript.
[2] The .dld file is a PostScript program downloaded to the printer.
[3] The ASCII driver allows you to generate ASCII files that do not contain printer escape codes.
[4] The Generic driver provides basic styling options across a wide range of printer models.

Box Characters	Bold	Italic	Under-line	Elite	Com-pressed	Letter Quality
No	Double	Yes	Dash	No	No	Yes
No	Yes	Yes	Yes	Yes	Yes	N/A
No	Double	Yes	Yes	Yes	Yes	Yes
No	Yes	Dash	Dash	Yes	Yes	Yes
No	Yes	Underline	Yes	Yes	Yes	N/A
No	Yes	Yes	Yes	Yes	Yes	Yes
No	Yes	Underline	Yes	Yes	Yes	No
No	Yes	Yes	Yes	Yes	Yes	No
Yes	Yes	Yes	Yes	Yes	Yes	Yes
Yes	Yes	Yes	Yes	Yes	Yes	Yes
No	Yes	Underline	Yes	Yes	Yes	No
No	Yes	Yes	Yes	Yes	Yes	No
No	Yes	Yes	Yes	Yes	Yes	Yes
No	Yes	Yes	Yes	Yes	Yes	Yes
No	Double	Underline	Dash	Yes	Yes	No
No	Double	Dash	Dash	No	Yes	Yes
Yes	Double	Dash	Dash	Yes	Yes	Yes
No	Yes	Underline	Yes	Yes	Yes	Yes
No	Yes	Underline	Yes	No	No	N/A
No	Yes	Yes	Yes	Yes	Yes	Yes
Yes	Yes	Yes	Yes	Yes	Yes	Yes
Yes	Yes	Underline	Yes	Yes	Yes	N/A
No	Yes	Yes	Yes	Yes	Yes	Yes
No	Double	Yes	Yes	Yes	Yes	Yes
No	Yes	Yes	Yes	Yes	Yes	Yes
No	Double	Dash	Dash	No	No	No
No	Yes	No	Yes	No	Yes	Yes
No	Yes	Yes	Yes	Yes	Yes	Yes
No	Double	Yes	Yes	Yes	Yes	Yes
No	Yes	Yes	Yes	Yes	Yes	Yes
No	Double	Yes	Yes	Yes	Yes	Yes
No	Yes	Yes	Yes	Yes	Yes	Yes
No	Yes	Yes	Yes	Yes	Yes	Yes
No	Yes	Underline	Yes	No	No	No
No	Yes	Underline	Yes	No	No	N/A
No	Yes	Underline	Yes	No	Yes	Yes
No	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	Yes	Yes
Yes	Double	Dash	Dash	No	No	No

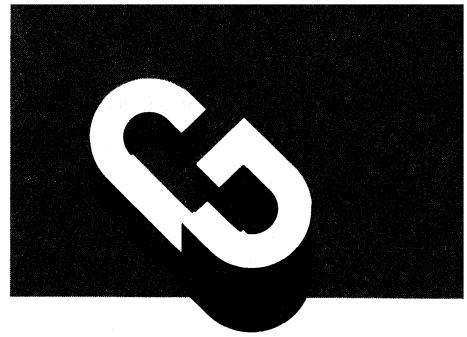
(continued)

Manufacturer	Model	Driver Filename
Hewlett-Packard, Inc.	Laserjet	
	LJ 6.5 lpi[5]	HPLAS100.pr2
	LJ 6 lpi[6]	HPLAS60.pr2
	LJ land 6.5 lpi[7]	HPLASL.pr2
	LJ land 6 lpi[8]	HPLASL45.pr2
	LJII 6.5 lpi, LJIID1 6.5 lpi (1-side)	HPLAS2I.pr2
	LJII 6 lpi, LJIID1 6 lpi (1-side) [6]	HPLAS260.pr2
	LJII land 6 lpi[8]	HPLS2L45.pr2
	LJIID2 6.5 lpi (2-side)	HPLASID.pr2
	LJIID2 6 lpi (2-side) [6]	HPLASD60.pr2
	LJIID1 6.5 lpi (1-side)	HPLAS2I.pr2
	DeskJet (HP2276)	HPDKS150.pr2
	PaintJet	HPPAINT.pr2
	QuietJet	HPQJET1.pr2
	ThinkJet	HPJET.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	RIC6K10.pr2
	6000 Landscape Orient	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.	P321, P341, P351	P351.pr2
	P1340, P1351	P1340.pr2
	PageLaser/2	TOSLAS15.pr2
Xerox Corporation	(Diablo) 630 API	DIAB630A.pr2

[5] lpi = lines per inch.
[6] Set page length to 60.
[7] Set page length to 51.
[8] Set page length to 45.

Box Characters	Bold	Italic	Under-line	Elite	Com-pressed	Letter Quality
No	Double	Underline	Yes	Yes	No	N/A
No	Double	Underline	Yes	Yes	No	N/A
No	Double	Underline	Yes	Yes	Yes	N/A
No	Double	Underline	Yes	Yes	Yes	N/A
Yes	Yes	Yes	Yes	Yes	Yes	N/A
Yes	Yes	Yes	Yes	Yes	Yes	N/A
Yes	Yes	Yes	Yes	Yes	Yes	N/A
Yes	Yes	Yes	Yes	Yes	Yes	N/A
Yes	Yes	Yes	Yes	Yes	Yes	N/A
Yes	Yes	Yes	Yes	Yes	Yes	N/A
No	Yes	Yes	Yes	Yes	Yes	N/A
No	Yes	Underline	Yes	Yes	Yes	N/A
No	Yes	Underline	Yes	Yes	Yes	No
No	Yes	Underline	Yes	No	Yes	No
Yes	Yes	Underline	Yes	No	Yes	Yes
Yes	Yes	Underline	Yes	No	Yes	Yes
Yes	Double	Underline	Yes	No	No	No
Yes	Double	Underline	Yes	No	No	No
Yes	Yes	Underline	Yes	Yes	Yes	Yes
Yes	Yes	Underline	Yes	Yes	Yes	Yes
No	Double	Underline	Yes	Yes	Yes	N/A
Yes	Yes	Underline	Yes	No	Yes	Yes
Yes	Yes	Yes	Dash	Yes	Yes	Yes
No	Double	Underline	Yes	Yes	Yes	N/A
No	Yes	Underline	Yes	No	No	N/A
No	Yes	Underline	Yes	Yes	Yes	N/A
No	Double	Underline	Yes	No	No	No
No	Double	Dash	Dash	No	Yes	No
No	Double	Underline	Yes	Yes	Yes	Yes
No	Double	Underline	Yes	Yes	Yes	Yes
Yes	Yes	Underline	Yes	No	Yes	Yes
No	Yes	Underline	Yes	Yes	Yes	Yes
Yes	Yes	Yes	Yes	Yes	Yes	Yes
Yes	Yes	Underline	Yes	No	Yes	Yes
Yes	Yes	Underline	Yes	No	Yes	Yes
No	Double	Underline	Yes	Yes	Yes	Yes
No	Yes	Underline	Yes	Yes	Yes	N/A
No	Yes	Underline	Yes	No	Yes	N/A
Yes	Yes	Underline	Yes	No	Yes	N/A
No	Yes	Yes	Yes	Yes	Yes	Yes
No	No	No	No	Yes	No	No
No	Yes	Yes	Dash	Yes	Yes	Yes
No	Yes	Yes	Yes	Yes	Yes	Yes
No	Double	Underline	Yes	Yes	Yes	Yes
Yes	Yes	Yes	Yes	No	Yes	N/A
No	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	'			

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

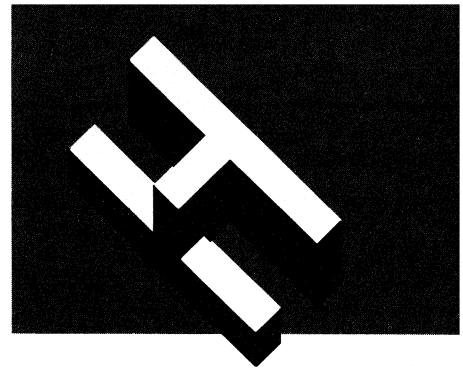
Binary	Hex	Decimal	Character
01100110	66	102	f
01100111	67	103	g
01101000	68	104	h
01101001	69	105	i
01101010	6A	106	j
01101011	6B	107	k
01101100	6C	108	l
01101101	6D	109	m
01101110	6E	110	n
01101111	6F	111	o
01110000	70	112	p
01110001	71	113	q
01110010	72	114	r
01110011	73	115	s
01110100	74	116	t
01110101	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

Binary	Hex	Decimal	Character
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	Binary	Hex	Decimal	Character
11000110	C6	198	╞	11100011	E3	227	π
11000111	C7	199	╟	11100100	E4	228	Σ
11001000	C8	200	╚	11100101	E5	229	σ
11001001	C9	201	╔	11100110	E6	230	µ
11001010	CA	202	╩	11100111	E7	231	τ
11001011	CB	203	╦	11101000	E8	232	Φ
11001100	CC	204	╠	11101001	E9	233	Θ
11001101	CD	205	═	11101010	EA	234	Ω
11001110	CE	206	╬	11101011	EB	235	δ
11001111	CF	207	╧	11101100	EC	236	∞
11010000	D0	208	╨	11101101	ED	237	ø
11010001	D1	209	╤	11101110	EE	238	∈
11010010	D2	210	╥	11101111	EF	239	∩
11010011	D3	211	╙	11110000	F0	240	≡
11010100	D4	212	╘	11110001	F1	241	±
11010101	D5	213	╒	11110010	F2	242	≥
11010110	D6	214	╓	11110011	F3	243	≤
11010111	D7	215	╫	11110100	F4	244	⌠
11011000	D8	216	╪	11110101	F5	245	⌡
11011001	D9	217	┘	11110110	F6	246	÷
11011010	DA	218	┌	11110111	F7	247	≈
11011011	DB	219	█	11111000	F8	248	°
11011100	DC	220	▄	11111001	F9	249	∙
11011101	DD	221	▌	11111010	FA	250	·
11011110	DE	222	▐	11111011	FB	251	√
11011111	DF	223	▀	11111100	FC	252	n
11100000	E0	224	α	11111101	FD	253	²
11100001	E1	225	ß	11111110	FE	254	■
11100010	E2	226	Γ	11111111	FF	255	

dBASE Commands and Functions Allowed in SQL Mode



The SQL language consists of a small set of commands and functions limited exclusively to the definition and access of data. dBASE IV supplements SQL commands with a set of dBASE commands and functions that you can use with SQL. In interactive mode, for example, you can use dBASE IV commands to create and print reports from data selected with SQL queries, or to BROWSE the results of a SELECT query.

The following sections list the commands and functions that you can use in SQL interactive and embedded modes.

dBASE Commands Allowed in SQL Mode

?/??
???
@
@...CLEAR
@...FILL
@...TO
ACCEPT
ACTIVATE MENU
ACTIVATE POPUP
ACTIVATE SCREEN
ACTIVATE WINDOW
APPEND
ASSIST
BEGIN/END TRANSACTION
BROWSE
CALL
CANCEL
CHANGE
CLEAR
CLEAR GETS
CLEAR MEMORY
CLEAR MENUS
CLEAR POPUPS
CLEAR TYPEAHEAD
CLEAR WINDOWS
CLOSE
CLOSE ALTERNATE
CLOSE FORMAT
CLOSE PROCEDURE
COMPILE
COPY FILE
CREATE
CREATE FROM
CREATE/MODIFY APPLICATION
CREATE/MODIFY LABEL
CREATE/MODIFY QUERY/VIEW
CREATE/MODIFY REPORT
CREATE/MODIFY SCREEN
CREATE/MODIFY STRUCTURE
CREATE VIEW FROM ENVIRONMENT
DEACTIVATE MENU
DEACTIVATE POPUP
DEACTIVATE WINDOW
DEBUG
DECLARE
DEFINE BAR
DEFINE BOX
DEFINE MENU
DEFINE PAD
DEFINE POPUP
DEFINE WINDOW
DELETE FILE
DIR
DO
DO CASE
DO WHILE
EDIT

EJECT
EJECT PAGE
ENDPRINTJOB
END TRANSACTION
ERASE
EXIT
FUNCTION
HELP
IF
INPUT
KEYBOARD
LABEL FORM
LIST/DISPLAY FILES
LIST/DISPLAY HISTORY
LIST/DISPLAY MEMORY
LIST/DISPLAY STATUS
LIST/DISPLAY STRUCTURE
LIST/DISPLAY USERS
LOAD
LOGOUT
MODIFY COMMAND/FILE
MOVE WINDOW
NOTE/*/&&
ON ERROR/ESCAPE/KEY
ON PAD
ON PAGE
ON READERROR
ON SELECTION PAD
ON SELECTION POPUP
PACK
PARAMETERS
PLAY MACRO
PRINTJOB
PRIVATE
PROCEDURE
PROTECT
PUBLIC
QUIT
READ
RECALL
RELEASE
RELEASE MENUS
RELEASE MODULE
RELEASE POPUPS
RELEASE WINDOWS
RENAME
REPORT FORM
RESTORE
RESTORE MACROS
RESTORE SCREEN
RESTORE WINDOW
RESUME
RETRY
RETURN
RUN/!
SAVE
SAVE MACROS
SAVE SCREEN
SAVE WINDOW
SELECT
SET
SET ALTERNATE
SET AUTOSAVE
SET BELL
SET BORDER
SET CATALOG
SET CENTURY
SET CLOCK
SET COLOR
SET CONFIRM
SET CONSOLE
SET CURRENCY
SET CURSOR
SET DATE
SET DBTRAP
SET DEBUG
SET DECIMALS
SET DEFAULT
SET DELETED
SET DELIMITERS
SET DESIGN
SET DEVELOPMENT
SET DEVICE
SET DIRECTORY
SET DISPLAY
SET ECHO
SET ENCRYPTION
SET ESCAPE
SET EXACT
SET EXCLUSIVE
SET FORMAT
SET FULLPATH
SET FUNCTION
SET HEADING
SET HELP
SET HISTORY
SET HOURS
SET INSTRUCT
SET INTENSITY
SET LOCK
SET MARGIN
SET MARK
SET MESSAGE
SET NEAR
SET ODOMETER
SET PATH
SET PAUSE
SET POINT

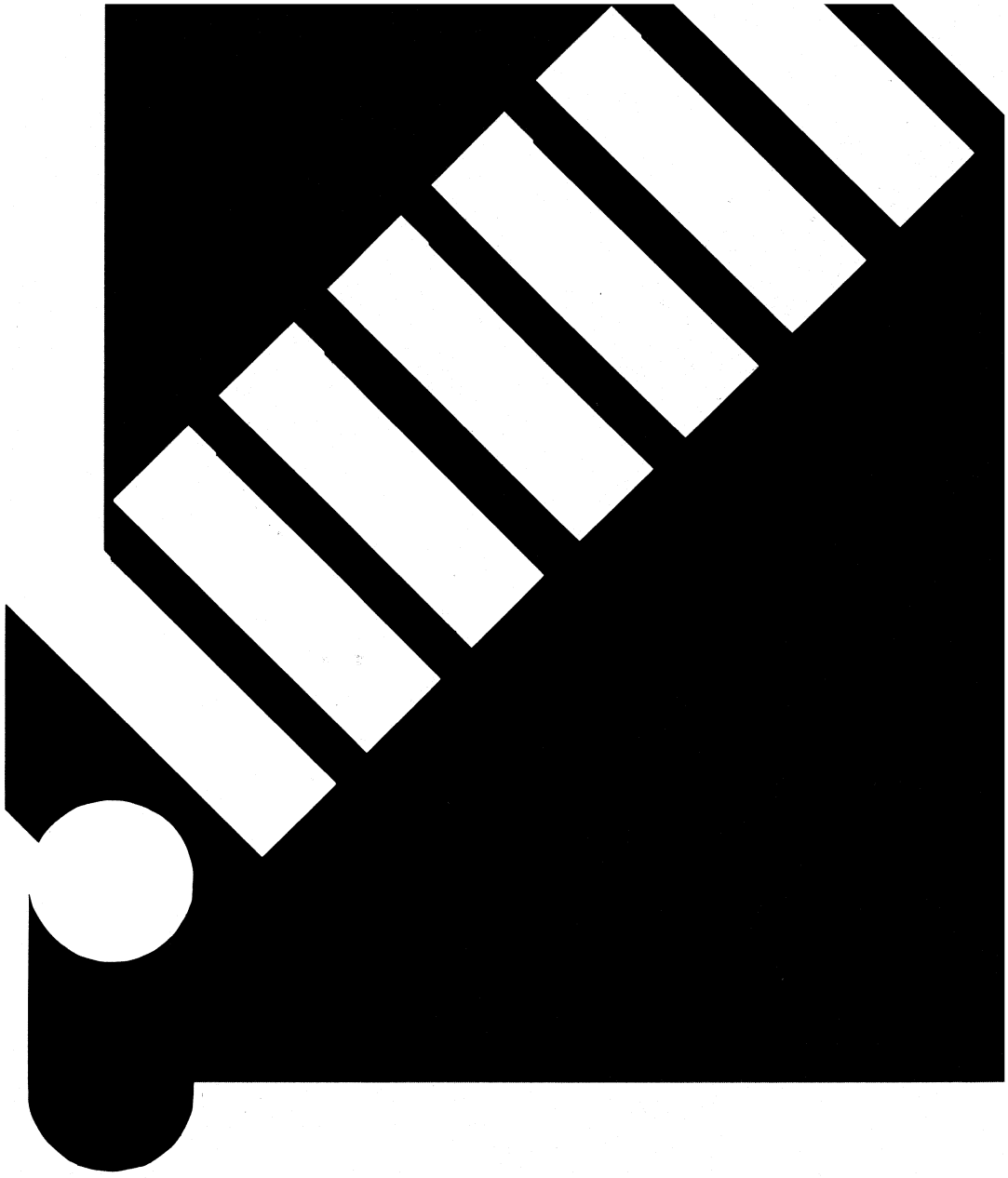
SET PRECISION
SET PRINTER
SET PROCEDURE
SET REFRESH
SET REPROCESS
SET SAFETY
SET SCOREBOARD
SET SEPARATOR
SET SPACE
SET SQL
SET STATUS
SET STEP
SET TALK
SET TITLE
SET TRAP
SET TYPEAHEAD
SET UNIQUE
SET VIEW TO
SET WINDOW OF MEMO TO
SHOW MENU
SHOW POPUP
SORT TO
STORE
SUSPEND
TEXT
TYPE
USE
WAIT

dBASE Functions Allowed in SQL Mode

NOTE
Certain dBASE functions may be allowed in SQL mode but not within a SQL command, for example, in a SELECT or WHERE clause. Such functions are noted with an asterisk.

&*
ABS()
ACOS()
ALIAS()*
ASC()
ASIN()
AT()
ATAN()
ATN2()
BAR()*
CALL()*
CDOW()
CEILING()
CERROR()*
CHR()
CMONTH()
COL()*
COMPLETED()*
COS()
CTOD()
DATE()
DAY()
DIFFERENCE()
DISKSPACE()*
DMY()
DOW()
DTOC()
DTOR()
DTOS()
ERROR()*
EXP()
FILE()*
FIXED()
FKLABEL()*
FKMAX()*
FLOAT()
FLOOR()
GETENV()*
IIF()*
INKEY()*
INT()
ISALPHA()*
ISCOLOR()*
ISLOWER()*
ISUPPER()*
KEY()*
LASTKEY()*
LEFT()
LEN()
LIKE()*
LINENO()*
LOG()
LOG10()
LOWER()
LTRIM()
MAX()*
MDY()
MEMORY()*

MENU()*
MESSAGE()*
MIN()*
MOD()
MONTH()
NETWORK()*
OS()*
PAD()*
PCOL()*
PI()
POPUP()*
PRINTSTATUS()*
PROGRAM()*
PROMPT()*
PROW()*
RAND()
READKEY()*
REPLICATE()
RIGHT()
ROLLBACK()*
ROUND()
ROW()*
RTOD()
RTRIM()
SELECT()*
SET()*
SIGN()
SIN()
SOUNDEX()
SPACE()
SQRT()
STR()
STUFF()
SUBSTR()
TAN()
TIME()
TRANSFORM()
TRIM()
TYPE()*
UPPER()
USER()
VAL()
VARREAD()*
VERSION()*
WINDOW()*
YEAR()



i 1 2 3 4 5 6 7 A B
C D E F G H In

Index

$ (Current display position), 4-99
& (Macro substitution), 4-1
* (Asterisk) wildcard, 4-71
* symbol, 1-11, 6-65, 6-78
. (Period) macro terminator, 4-1
? (Question mark) wildcard, 4-71
? symbol, 1-11
?/??
    indenting paragraphs, 5-5
    left margin definition, 5-6
    output alignment, 5-1
    paragraph indent, 5-5
    right margin definition, 5-35
    setting tabs, 5-37
    with ON ESCAPE, 2-3
    with ON PAGE, 2-3
    with SET ALTERNATE, 2-1
    with SET CONSOLE, 2-1
    with SET PRINTER, 2-1
???
    for Postscript printers, 2-8
    printer control codes, 2-6
    to change document styles, 2-4
@, 1-20
    $ function for current row and column, 2-11
    $ template for currency format, 2-14, 2-15
    COLOR, 2-11
    DEFAULT, 2-12
    ERROR, 2-12
    format functions, 2-14, 2-15
    GET, 2-11
    PICTURE, 2-13
    RANGE, 2-13
    REQUIRED, 2-14
    SAY, 2-11
    VALID, 2-13
    WINDOW, 2-12
    with SET DEVICE TO FILE, 2-10
    with SET DEVICE TO PRINTER, 2-10
@...CLEAR
    to erase parts of display, 2-20
@...FILL
    to color the display, 2-21
@...TO
    box–drawing characters, 2-23
    color attributes, 2-23
    for box drawing on screen, 2-22
    in a format file, 2-23

A

ABS(), 4-3
ACCEPT, 1-10
    escape, 2-24
    maximum characters, 2-24
Access level, user, 4-4
ACCESS(), 4-4
ACOS(), 4-6
ACTIVATE MENU, 2-25
ACTIVATE POPUP, 2-26
ACTIVATE SCREEN, 2-27
ACTIVATE WINDOW, 2-28
Adding records to a database file, 2-29
ADDITIVE, 3-2, 3-3
Aggregate functions (SQL)
    in SELECT clause, 6-65, 6-66
Alias
    as name, 1-6
    as work area letter, 1-6
    as work area number, 1-6
    indirect reference to, 1-7
    restricted characters, 1-6
    to qualify field name, 1-11
    with IN, 1-11
Alias (SQL)
    defining, 6-67
    in SQL SELECT clause, 6-65
Alias name restrictions, 2-277
ALIAS(), 4-7
Alignment of output, 5-1, 5-2

ALL keyword, 1-10, 1-11
ALL keyword (SQL)
in GRANT, 6-48
in REVOKE, 6-59
in SELECT, 6-65
Alphabetical characters, checking for, 4-62
Alt key combinations (table), 4-60
ALTER privilege (SQL), 6-48
ALTER TABLE (SQL), 6-6, 6-63, 7-5
Alternate file, 3-2
Amortization of loan, 4-98
Angles
arccosine, 4-6
arcsine, 4-9
arctangent, 4-11, 4-12
converting degrees to radians, 4-37
converting radians to degrees, 4-119
cosine, 4-26
sine, 4-129
tangent, 4-138
APPEND, 1-19, 2-29, 3-4
APPEND FROM, 2-31, 3-4
APPEND FROM ARRAY, 2-35
APPEND MEMO, 2-27
Appending
ASCII files, 2-32
Framework files, 2-32
Lotus 1-2-3 files, 2-33
SYLK files, 2-33
Applications Generator objects, 2-82
Arccosine, 4-6
Arcsine, 4-9
Arctangent, 4-11, 4-12
Array
as memory variables, 2-105
definition list, 2-104
element names, 2-105
maximum size, 2-104
names, 2-104
PRIVATE, 2-104
PUBLIC, 2-104
Array elements
in expression, 1-11
use, 1-11
ASC(), 4-8
ASCII chart , G-1
ASCII files
decimal value of character, 4-8
evaluating ASCII values, 6-69
exporting to from SQL, 6-89
generating, 5-19
importing to SQL from, 6-54, 6-55
ASIN(), 4-9
Assembly program preparation, 2-182
ASSIST, 2-39
AT(), 4-10
ATAN(), 4-11
ATN2(), 4-12
Available memory, 2-173, 2-174
AVERAGE, 1-10, 2-40

B

Backup filename extensions, 2-188
Backward compatibility
binary program files, 2-181
.dbo files, 2-59
format files, 2-91
labels, 2-85
.ndx indexes, 2-275
object files, 2-125
report forms, 2-89, 2-232
SET PROCEDURE TO, 2-126
structure extended files, 2-75
view files, 2-86, 2-96
BAR
MESSAGE placement, 2-107
selection conditional, 2-107
selection denied, 2-107
Bar menu pads, 2-112
BAR(), 4-13
BEGIN/END TRANSACTION, 2-41
with DELETE, 6-36
with ROLLBACK, 6-61
Beginning of file, 4-14
Binary Coded Decimal numbers, 1-13
converting, 4-44, 4-47
Binary program files, 4-15
Binary program loading, 2-181
Blanks
making character strings, 4-132
removing, 4-80, 4-120
BOF(), 4-14
_box variable, 2-109
Boxes
controlling display of, 5-3
controlling height of, 5-33
Branching
on ON ERROR, 2-193
on ON ESCAPE, 2-193
Breakpoints, 4-72
BROWSE, 1-19, 3-4
calculated fields, 2-48
format files, 2-46
index files, 2-45
menu bar, 2-46
options, 2-46, 2-47
read only, 2-48

C

CALCULATE, 1-10, 2-50
CALL
  LOAD, 2-53
  parameter options, 2-53
CALL(), 4-15
Calling binary programs, 2-53
CANCEL, 2-54
  as ROLLBACK, 2-42
.cat files, 1-5
Catalog tables (SQL)
  creating entries, 6-11, 6-28
  described, 7-4
  displaying information, 7-2
  effect of DROP DATABASE, 6-38
  effect of DROP TABLE, 6-41
  object definitions, 7-3
  statistics, 6-63, 7-3
  tracking privileges, 6-47
  updating, 7-3
  verifying entries, 6-26
Catalog, dBASE
  effect of CREATE TABLE, 6-18
  effect of DROP DATABASE, 6-38
  effect of DROP TABLE, 6-41
  effect of DROP VIEW, 6-42
  effect of SAVE TO TEMP, 6-75
  effect of START DATABASE, 6-85
CDOW(), 4-16
CEILING(), 4-17
CERROR(), 4-18
CHANGE. *See* EDIT
CHANGE(), 4-20
Character data type, 1-12
Character string functions
  ASC(), 4-8
  AT(), 4-10
  CHR(), 4-21
  DIFFERENCE(), 4-32
  IIF(), 4-54
  ISALPHA(), 4-62
  ISLOWER(), 4-64
  ISUPPER(), 4-66
  LEFT(), 4-69
  LEN(), 4-70
  LIKE(), 4-71
  LOOKUP(), 4-78
  LOWER(), 4-79
  LTRIM(), 4-80
  REPLICATE(), 4-113
  RIGHT(), 4-114
  RTRIM(), 4-120
  SOUNDEX(), 4-130
  SPACE(), 4-132
  STUFF(), 4-135
  SUBSTR(), 4-136
  TRANSFORM(), 4-140
  TRIM(), 4-141
  UPPER(), 4-144
  VAL(), 4-146
Character strings
  comparisons, 4-71
  converting date to, 4-36
  converting from number, 4-21
  converting numeric values, 4-134
  converting to date, 4-27
  converting to lowercase, 4-79
  converting to numbers, 4-146
  converting to uppercase, 4-144
  extracting substrings, 4-136
  formatting, 4-140
  of spaces, 4-132
  removing blanks, 4-120
  removing leading blanks, 4-80
  replacing part, 4-135
Characters, repeating, 4-113
CHR(), 1-12, 4-21
CLEAR
  ALL, 2-56
  MEMORY, 2-57
  MENUS, 2-57
  options, 2-56, 2-57
  SCREENS, 2-57
  TYPEAHEAD, 2-57
  WINDOWS, 2-57
  without options, 2-56
CLOSE (SQL), 6-9, 6-43
CLOSE, 2-58
CMONTH(), 4-23
COL(), 4-24
Color attributes, 3-1
Column position, 4-24, 4-99
Columns
  adding, 6-6
  changing values, 6-91
  in a view, 6-22
  initializing values, 6-51
  INSERTing values, 6-51
  name qualification, 6-65
  naming, 6-20
  removing, 6-7
  restrictions, 6-7
  SELECT headings, 6-66
  SELECTing, 6-65
Command line
  components, 1-9
  constructing, 1-1

defined, 1-1
specifying drive, 1-5
specifying path, 1-5
Command verb, 1-1
Commands (SQL)
allowed in transaction, 6-61
classes of, 6-3
data definition, 6-4
database creation, 6-3
database security, 6-3
database startup, 6-3
deletion, 6-4
embedded, 6-5
entering, 1-3
object creation, 6-3
object modification, 6-3
query update, 6-5
syntax, 6-1
UDF restrictions, 1-19
utility, 6-5
Commands, dBASE
abbreviating, 1-3
defined, 1-1
entering, 1-2, 1-3
not case sensitive, 1-3
notation, 1-2
on command line, 1-1
re-entering, 1-3
use in SQL mode, 1-3, 4-1
Common logarithm, 4-77
Comparison operators (SQL), 6-68
COMPILE, 1-9
code optimization, 2-61
DO, 2-60
RUNTIME option, 2-59
COMPLETED(), 2-42, 4-25
COMSPEC, 4-53
Concatenation operator, 1-15
Condition
as logical expression, 1-14
in syntax, 1-10
Conditional command processing, 2-151
Config.db file
changing date format, 1-13
changing history buffer, 1-3
changing system parameters, 1-16
setting DBTRAP, 1-19
SQLDATABASE command, 6-12
SQLHOME command, 6-11, 6-84
Constant
defined, 1-12
in expression, 1-11
in SQL SELECT clause, 6-66
CONTINUE, 2-62
Controlling index, 2-155
Conversion functions
& (Macro substitution), 4-1
ASC(), 4-8
CHR(), 4-21
CTOD(), 4-27
DMY(), 4-34
DTOC(), 4-36
DTOR(), 4-37
DTOS(), 4-38
FIXED(), 4-44
FLOAT(), 4-47
LOWER(), 4-79
MDY(), 4-84
RTOD(), 4-119
STR(), 4-134
TRANSFORM(), 4-140
UPPER(), 4-144
VAL(), 4-146
Conversion of data types, 2-95
CONVERT, 2-63, 4-20, 4-73
dbaselock field, 2-63
LKSYS(), 2-63, 4-73
Converting
.dbf to an SQL table, 2-95
fields character to logical, 2-95
fields date to character, 2-95
fields logical to character, 2-95
fields numeric to character, 2-95
.mdx tags to .ndx files, 2-76
COPY, 2-65
with CONVERT, 2-66
with PROTECT, 2-66
with SET RELATION, 2-67
COPY FILE, 2-69
COPY INDEXES, 2-70
COPY MEMO, 2-72
COPY STRUCTURE, 2-73
COPY STRUCTURE EXTENDED, 2-75
COPY TAG, 2-76
COPY TO ARRAY
FOR clause, 2-77
SET FIELDS, 2-77
WHILE clause, 2-77
Correlated subquery (SQL), 6-67, 6-72, 6-78
COS(), 4-26
Cosine, 4-26
COUNT, 1-10
with SET FILTER TO, 2-79
with SET LOCK, 2-79
CREATE DATABASE (SQL), 6-9, 6-11, 6-85, 7-1, 7-6

CREATE FROM, 2-80
CREATE INDEX (SQL), 6-13, 6-63, 7-3, 7-6
CREATE REPORT, 1-9
CREATE SYNONYM (SQL), 6-16
CREATE TABLE (SQL), 6-18
CREATE VIEW (SQL), 6-21, 6-78
CREATE VIEW FROM ENVIRONMENT, 1-4, 2-96
CREATE/MODIFY APPLICATION, 2-82
CREATE/MODIFY LABEL, 1-4, 2-84
CREATE/MODIFY QUERY/VIEW, 1-4, 2-86
, CREATE/MODIFY REPORT, 1-4, 2-88
CREATE/MODIFY SCREEN, 2-90
CREATE/MODIFY STRUCTURE, 2-92
CTOD(), 1-13, 4-27
Cursor (SQL)
  activating, 6-56
  deactivating, 6-9
  defining, 6-31
  dropping, 6-87
  FETCH positioning, 6-43
  referencing, 6-32
  using to delete, 6-35, 6-36
  using to update, 6-74, 6-92
Cursor position
  column, 4-24
  row, 4-118
Custom screen forms, 2-90
Cut-sheet printing, 5-34
.cvt extension, 2-64

D

Data encryption, 2-215
Data type of an expression, 4-142
Data types (SQL), 6-7, 6-19, 6-68
Data types, dBASE
  abbreviations for, 1-12
  character type, 1-12
  compatibility within expression, 1-12
  date type, 1-13
  in expression list, 1-12
  logical type, 1-14
  numeric type, 1-13
Database (SQL)
  activating, 6-85
  closing, 6-85, 6-87
  creating, 6-11
  default, 6-85
  deleting, 6-38
  displaying information, 6-84
  naming, 6-11
Database design screen, 2-92
Database directory (SQL)
  creating, 6-11
  defining existing, 6-11
Database file
  design, 2-92
  exporting to from SQL, 6-89
  importing to SQL from, 6-54
  name, 4-30
  SQL table as, 6-18
  structure, E-1
Database file functions
  BOF(), 4-14
  DELETED(), 4-31
  EOF(), 4-39
  FOUND(), 4-51
  LOOKUP(), 4-78
  LUPDATE(), 4-81
  MEMLINES(), 4-85
  MLINE(), 4-90
  RECCOUNT(), 4-110
  RECNO(), 4-111
  RECSIZE(), 4-112
  SEEK(), 4-122
Date
  converting, 4-27, 4-36, 4-38
  data type, 1-13
  day of month, 4-29
  day of week, 4-16, 4-35
  determining earlier, 4-89
  determining later, 4-82
  determining month, 4-92
  determining year, 4-151
  formatting, 4-34, 4-84, 4-140
  indexing, 4-38
  month, 4-23
  of update, 4-81
  system, 4-28
Date functions
  CDOW(), 4-16
  CMONTH(), 4-23
  CTOD(), 4-27
  DATE(), 4-28
  DAY(), 4-29
  DMY(), 4-34
  DOW(), 4-35
  DTOC(), 4-36
  DTOS(), 4-38
  MDY(), 4-84
  MONTH(), 4-92
  YEAR(), 4-151
DATE(), 4-28
Dates
  entering blank, 1-13
  subtracting, 1-13

Day of month, 4-29
Day of week, 4-16, 4-35
DAY(), 4-29
dBASE
  search order for procedures, 2-210
  text editor, 2-187
dBASE II files
  exporting to from SQL, 6-89
  importing to SQL from, 6-54
dBASE III PLUS labels, 2-85
dBASE IV version number, 4-149
dBASE mode
  entering SQL mode, 1-3
dBASE program file. *See* .prg file
_dbaselock
  field contents, 2-63
DBCHECK (SQL), 6-26
DBDEFINE (SQL), 6-28, 6-75, 7-3
DBDEFINE, 2-95
DBF(), 4-30
DBLINK, 1-9
.dbo file
  backward compatibility, 2-59
  creation, 1-7
  procedure list, 1-8, 2-60
Dbsystem.db file, 2-214
DEACTIVATE MENU, 2-98
DEACTIVATE POPUP, 2-99
DEACTIVATE WINDOW, 2-100
DEBUG, 2-101
  breakpoint window, 2-101
  commands, 2-102
  display window, 2-101
  edit window, 2-101
  parameter list, 2-103
  RESUME, 2-102
  toggle keys, 2-102
  windows, 2-101
DECLARE, 1-11
  for arrays, 2-104
DECLARE CURSOR (SQL), 6-9, 6-31, 6-35, 6-43, 6-56, 6-74, 6-80
DEFINE BAR of a pop-up menu, 2-107
DEFINE BOX, 2-109
DEFINE MENU, 2-111
DEFINE PAD, 2-112
DEFINE POPUP, 2-114
DEFINE WINDOW, 2-116
Degrees, 4-119
  to radians, 4-37
DELETE, 2-118
DELETE (SQL), 6-34, 6-35, 6-36, 6-37, 6-63, 6-78
  embedded form, 6-35, 6-36, 6-80
  using cursor, 6-32
  with WHERE clause, 6-35
DELETE FILE. *See* ERASE
DELETE privilege, 6-48
DELETE TAG, 2-120
DELETED(), 4-31
Deleting
  all records, 2-280
  files, 2-139
  marked records, 2-118, 2-204
  .mdx tags, 2-120
Delimiters
  to define character constant, 1-12
  to define date constant, 1-13
DIFFERENCE(), 4-32
DIR, 2-122
Disk menu, 3-1
Disk space, 4-33
DISKSPACE(), 4-33
Display
  checking color capabilities, 4-63
  determining column, 4-24
  determining row, 4-118
DISPLAY, 1-19, 2-124
Display attributes, 3-1
Display functions
  COL(), 4-24
  ISCOLOR(), 4-63
  ROW(), 4-118
DISPLAY HISTORY, 1-3
Display menu, 3-1
DISTINCT keyword (SQL), 6-66
DMY(), 4-34
DO, 1-7, 1-9
  search order, 2-125
  SET PROCEDURE TO, 2-125
  with update queries, 2-128
DO CASE
  versus IF, 2-129, 2-152
DO WHILE, 2-131
DOS environment, 4-53
Dot prompt
  as command line, 1-1
  entering SQL mode, 1-3
  redisplaying commands at, 1-3
  signals interactive mode, 1-2
DOW(), 4-35
Draft print, 5-30
Drive specification
  explicit, 1-5
  saving, 1-5, 1-6
Drivers, printer
  activating, 5-18
DROP DATABASE (SQL), 6-38

DROP INDEX (SQL), 6-39
DROP SYNONYM (SQL), 6-40
DROP TABLE (SQL), 6-41
DROP VIEW (SQL), 6-42
DTOC(), 4-36
DTOR(), 4-37
DTOS(), 4-38

E

e constant, powers of, 4-41
EDIT, 2-133
    APPEND, 2-133
    control keys, 2-133
    in UDFs, 1-19
    navigation keys, 2-134, 2-135
    options NOAPPEND, 2-134
    options NOCLEAR, 2-134
    options NODELETE, 2-134
    options NOEDIT, 2-134
    options NOFOLLOW, 2-134
    options NOINIT, 2-134
Editing view files, 2-87
Editing window
    entering dBASE commands, 1-3
    entering SQL commands, 1-3
    opening, 1-3
EJECT, 2-136
EJECT PAGE, 2-137
Encryption of index files, 2-216
Encryption, dBASE, 2-215
    effect on DBCHECK, 6-26
    use of DBDEFINE, 6-29
Encryption, SQL
    during LOAD DATA, 6-55
    use of DBDEFINE, 6-29
END TRANSACTION, 4-25
ENDCASE, 2-129
ENDDO, 2-131
ENDIF, 2-151
Environment system variables, 4-53
EOF(), 4-39
ERASE FILE, 2-139
Erasing memo files, 2-139
ERROR(), 4-40
Errors
    locating, 4-103
    messages, 4-88, A-1
    numbers, 4-40
EXCEPT keyword, 1-11
Exclusive use
    for DELETE TAG, 2-120
    for INDEX, 2-157
    for MODIFY STRUCTURE, 2-94
    required with RESET, 2-234
    required with ZAP, 2-280
Executable object code files, 2-59
EXIT, 2-131
EXP(), 1-13, 4-41
EXPORT, 2-140
Export file types (SQL), 6-89
Exporting data with COPY, 2-65
Exporting SQL data, 6-88
Expression list
    data type and, 1-12
Expressions
    components, 1-11
    in SELECT clause, 6-65, 6-66
    in SQL search condition, 6-68

F

FETCH (SQL), 6-10, 6-31, 6-36, 6-43, 6-44, 6-45, 6-56
Field name
    defined, 1-11
    in expression, 1-11
    qualifying, 1-11
Field test functions
    DIFFERENCE(), 4-32
    IIF(), 4-54
    ISALPHA(), 4-62
    ISLOWER(), 4-64
    ISUPPER(), 4-66
    LEN(), 4-70
    SOUNDEX(), 4-130
    TYPE(), 4-142
FIELD(), 4-42
Fields
    as production .mdx tags, 2-93
    maximum number per record, 2-93
    names, 2-93
    size limits in exported files, 2-140
    types, 2-93
File
    exporting options, 2-66
    extensions, D-1
    formats for export, 2-66
    privilege schemes, 2-215
    privileges, changing, 2-215
FILE(), 4-43
Filename
    drive specification, 1-5
    extension, 1-4, 1-5
    indirect reference to, 1-4
    path specification, 1-4, 1-5

saving, 1-5
specifying, 1-4
Files
date of update, 4-81
determining existence, 4-43
determining name, 4-30
locked, 4-73
locking, 4-48
menu, 3-1
FILES option of popup, 2-115
Financial functions
FV(), 4-52
PAYMENT(), 4-98
PV(), 4-106
FIND
index files, 2-141
LOCATE differences, 2-141
SEEK differences, 2-141
FIXED(), 4-44
FKLABEL(), 4-45
FKMAX(), 4-46
FLOAT(), 1-13, 4-47
Floating point numbers, 1-13
converting, 4-44, 4-47
FLOCK(), 4-48
FLOOR(), 4-50
.fmo file, 1-9
Footers with ON PAGE, 2-198
FOR clause, 1-10
FOR UPDATE OF clause (SQL), 6-32, 6-74, 6-76
Format files, 2-90
double quotes in, 2-91
Formatting
character strings, 4-140
dates, 4-34, 4-84
Formfeed, 2-136. *See also* EJECT
with Postscript printer, 2-8
Forms design screen, 2-90
FOUND(), 4-51
Framework II files
exporting to from SQL, 6-89
Framework II files
importing to SQL from, 6-54
.frg file, 1-9
.frm file, 1-5
.fro file, 1-9
FROM clause (SQL), 6-65, 6-67
Full-screen commands
current field or variable, 4-147
exiting, 4-108
Function keys, 4-45, 4-46
INKEY() values (table), 4-57
FUNCTION, 1-17, 2-144
Functions, dBASE, 4-1
in expression, 1-11
in INSERT, 6-51
in search condition, 6-68, 6-70
in SELECT clause, 6-66
macro substitution, 1-16, 1-17
notation, 1-2
on command line, 1-1
operation, 1-2
use in SQL mode, 1-3
using, 1-16
Future value, 4-52
FV(), 4-52

G

Generic.pr2 driver, 5-19
GETENV(), 4-53
GO/GOTO
access denied in related files, 2-149
IN clause, 2-148
in related files, 2-148
with SET DELETED, 2-148
Going from EDIT to BROWSE, 2-133
GRANT (SQL), 6-46, 6-47, 6-48, 6-49, 6-58, 7-4, 7-5
GROUP BY clause (SQL), 6-72, 6-78
use of index, 6-13
Group result (SQL), 6-72

H

HAVING clause (SQL), 6-72, 6-78
HELP, 2-150
History buffer, 1-3

I

IBM ASCII character set, 3-8
Identification functions
ALIAS(), 4-7
BAR(), 4-13
DBF(), 4-30
FIELD(), 4-42
FILE(), 4-43
FKLABEL(), 4-45
FKMAX(), 4-46
GETENV(), 4-53
MDX(), 4-83
NDX(), 4-93
NETWORK(), 4-94
ORDER(), 4-95

OS(), 4-96
PAD(), 4-97
PROGRAM(), 4-103
PROMPT(), 4-104
SET(), 4-125
TAG(), 4-137
VARREAD(), 4-147
VERSION(), 4-149
IF, 2-151
versus DO CASE, 2-129, 2-152
IF...ELSE
with DELETE, 6-36
IIF(), 4-54
Immediate IF, 4-54
IMPORT
supported formats, 2-153
Import file types (SQL), 6-54
Importing data (SQL), 6-53
IN keyword, 1-11
Indent, 5-5
paragraph of ? output, 5-5
_indent, 5-5
INDEX, 2-154
Index (SQL)
automatic operation, 6-14
catalog entry, 7-1, 7-7
creating, 6-13
deleting, 6-39
effect of DROP TABLE, 6-41
naming, 6-13
restrictions, 6-13
unique, 6-14
updating statistics, 6-63
use of DBDEFINE, 6-28
Index files
determining name, 4-83, 4-93
encryption, 2-216
key expression, 4-67
searching, 4-51, 4-122
using dates, 4-38
Index keys and data type matching, 2-155
Index, length of conditional, 2-155
INDEX privilege (SQL), 6-48
Index tag (SQL), 6-14
Index, dBASE
UNIQUE, 6-28
with SET DELETED, 2-157
with SET FILTER, 2-157
with transactions, 2-157
Indexing
conditional, 2-155
data type conversions, 2-156
.mdx file updates, 2-156
on memory variables, 2-157
on UDFs, 2-147
SET ORDER, 2-156
variable length keys, 2-156
with UNIQUE, 2-156
Indirect file referencing
in lieu of macro substitution, 1-17
using, 1-4, 1-11
INKEY(), 4-56, 4-57
Inner query. *See* Subquery
INPUT, 1-10, 2-160
Input functions
INKEY(), 4-56
LASTKEY(), 4-68
READKEY(), 4-108
Input, streaming
positioning, 1-20
system memvars affecting, 1-20
INSERT, 2-161
INSERT (SQL), 6-50, 6-51, 6-52, 6-63, 6-78
INSERT privilege (SQL), 6-48
INT(), 4-61
Integer, 4-17, 4-50, 4-61
Interest rate, 4-52, 4-98, 4-106
Interrupts in UDFs, 2-145
INTO clause (SQL), 6-67, 6-79
ISALPHA(), 4-62
ISCOLOR(), 4-63
ISLOWER(), 4-64
ISMARKED(), 4-65
ISUPPER(), 4-66

J

Join (SQL), 6-78
JOIN, 2-163

K

KEEP option (SQL), 6-75
Key expression, 4-67
Key label names, 2-194, 2-195
KEY(), 4-67
KEYBOARD
special keys, 2-165
with SET TYPEAHEAD, 2-165
Keyboard macro execution, 2-206
Keys menu, 3-1
Keystrokes, 4-56, 4-68, 4-108
Keywords, dBASE
abbreviating, 1-3

L

Label
  designer, 2-84
  format file, 2-167
  template, 2-167
LABEL FORM
  file locking, 2-168
  printer formatted file, 2-168
  with SET DEVELOPMENT, 2-167
Labels design screen,
  to change system memvars, 1-21
LASTKEY(), 4-68
.lbl files, 1-5
.lbo files, 1-9, 2-85
LEFT(), 4-69
LEN(), 4-70
Letter-quality print, 5-30
LIKE keyword, 1-11
LIKE(), 4-71
Line
  numbers, 4-72
  spacing, 5-33
LINENO(), 4-72
LIST, 1-19
LIST column headings, 2-170
LIST HISTORY, 1-3
LIST TO PRINTER, 2-169
LIST/DISPLAY, 2-169
LIST/DISPLAY FILES, 2-171
LIST/DISPLAY HISTORY, 2-172
LIST/DISPLAY MEMORY, 2-173
LIST/DISPLAY STATUS, 2-176
LIST/DISPLAY STRUCTURE, 2-178
LIST/DISPLAY USERS, 2-180
Literal text strings in programs, 2-268
LKSYS(), 4-73
_lmargin, 5-6, 5-7
LOAD
  restrictions, 2-181, 2-182
LOAD DATA (SQL), 6-53, 6-54, 6-55
LOCATE, 2-184, 4-51
  CONTINUE, 2-184
  FOR, 2-184
  scope, 2-184
LOCK(), 4-75, 4-115
Locked records and files, 4-73
Locking
  files, 4-48, 4-73
  records, 4-73, 4-115
LOG(), 1-13. 4-76
Log-in name, 4-73, 4-145
Logarithm, 4-41, 4-76, 4-77
Logical data type, 1-14
Logical expression, 1-14
Logical operators
  in SQL search condition, 6-69
  listed, 1-14
  precedence, 1-15
Logical values, formatting, 4-140
LOGOUT, 2-186
LOG10(), 4-77
LOOKUP(), 4-78
Lookups, in indexed files, 4-78, 4-122
LOOP, 2-131
Lotus files
  exporting to from SQL, 6-89
  importing to SQL from, 6-54
LOWER(), 4-79
Lowercase
  checking for, 4-64
  conversion, 4-79
LTRIM(), 4-80
LUPDATE(), 4-81

M

Macro
  in DO WHILE, 2-131, 2-132
  substitution function, 1-16, 1-17
  substitution in UDFs, 2-147
  terminator, 4-1
Margin
  aligning text between, 5-1
  defining left, 5-6
  defining right, 5-35
Marking records for deletion, 2-118
Math functions
  ABS(), 4-3
  CEILING(), 4-17
  EXP(), 4-41
  FIXED(), 4-44
  FLOOR(), 4-50
  FV(), 4-52
  INT(), 4-61
  LOG(), 4-76
  LOG10(), 4-77
  MAX(), 4-82
  MIN(), 4-89
  MOD(), 4-91
  PAYMENT(), 4-98
  PI(), 4-100
  PV(), 4-106
  RAND(), 4-107
  ROUND(), 4-117
  SIGN(), 4-128
  SQRT(), 4-133

Mathematical operators
listed, 1-14
precedence, 1-15
MAX(), 4-82
Maximum character field size, 2-93
Maximum fields per record, 2-93
Maximum length
of conditional index expression, 2-155
of evaluated index key, 2-155
of index key expression, 2-155
Maximum memo field entry, 2-93
Maximum number
of active .dbo files, 2-126
of DO WHILEs, 2-131
of GETS, 2-56
of LOAD files, 2-181
of memory variables, 2-263
of nested DOs, 2-127
of open files, 2-276
of procedures in a program, 2-210
of SORT fields, 2-260
of user–defined functions, 2-144
Maximum numeric field size, 2-93
Maximum record size, 2-93
Maximum size of arrays, 2-104
Maximum value, 4-82
.mdx files, 4-83, 4-137
MDX(), 4-83
MDY(), 4-84
MEMLINES(), 4-85
Memo field
disallowed in SQL, 6-7
effect of DBDEFINE, 6-28
extracting lines, 4-90
extracting substrings, 4-69, 4-114, 4-136
number of lines, 4-85
replacing part, 4-135
Memory variable, system
changing defaults, 1-20
defined, 1-2, 1-19
in expression, 1-11
notation, 1-2
on command line, 1-1
operation, 1-2
paragraph-specific, 1-20
printer-specific, 1-20
printjob-specific, 1-20
streaming input and, 1-20
using, 1-19, 1-20, 1-21
Memory variable, temporary. *See* Memvar
MEMORY(), 4-86
Memory, determining capacity, 4-86
MEMOWIDTH, 4-85
Memvar
changing precedence, 1-12
commands used to create, 1-10
created by FETCH, 6-43
editing with READ, 2-220
entering row values to, 6-79
FETCH processing, 6-31, 6-43
in array, 1-11
in expression, 1-11
in search condition, 6-68
in SELECT clause, 6-66
INSERTing, 6-51
names, 2-264
purpose, 1-10
Memvar test functions
DIFFERENCE(), 4-32
IIF(), 4-54
ISALPHA(), 4-62
ISLOWER(), 4-64
ISUPPER(), 4-66
LEN(), 4-70
SOUNDEX(), 4-130
TYPE(), 4-142
Menu
choices, 2-108
deactivating, 2-98
defining, 2-111
definition command sequence, 2-197
definition sequence, 2-113
determining name of, 4-87
determining pad name, 4-97
determining prompt name, 4-104
MESSAGE overwritten by PAD, 2-111
MESSAGE placement, 2-111
Menu functions
BAR(), 4-13
MENU(), 4-87
PAD(), 4-97
POPUP(), 4-101
PROMPT(), 4-104
MENU(), 4-87
Merging records from two files, 2-163
MESSAGE
DEFINE BAR, 2-107
DEFINE PAD, 2-112
MESSAGE(), 4-88
MIN(), 4-89
Minimum value, 4-89
MLINE(), 4-90
MOD(), 4-91
MODIFY, 3-2, 3-3
MODIFY COMMAND/FILE, 2-187

MODIFY STRUCTURE
backup file, 2-94
in SQL, 2-95
Month, 4-23, 4-92
MONTH(), 4-92
MOVE WINDOW, 2-190
in UDFs, 2-145
Multiplan files
exporting to from SQL, 6-89
importing to SQL from, 6-54
Multiple index
determining name, 4-83
effect of DBDEFINE, 6-28
file, 2-155
key expression, 4-67
tag name, 4-137
MVBLKSIZE, 2-173
MVMAXBLKS, 2-173

N

Name of database file, 4-30
Name, alternative. *See* Synonym (SQL)
Names of arrays, 2-104
.ndx files, 2-155, 4-137
NDX(), 4-93
Network functions
ACCESS(), 4-4
CHANGE(), 4-20
FLOCK(), 4-48
LKSYS(), 4-73
NETWORK(), 4-94
RLOCK()/LOCK(), 4-115
USER(), 4-145
NETWORK(), 4-94
NEXT keyword, 1-10
NOTE, 2-191
Numeric data type, 1-13
Numeric values
converting, 4-44, 4-47
converting character strings, 4-146
converting to character string, 4-134
determining sign, 4-128
formatting, 4-140
rounding, 4-117

O

Object code, 1-7
Object file extension requirement, 2-127
Objects (SQL), 6-11, 7-1
ON ERROR
operating system errors, 2-193
restriction in full–screen commands, 2-192
ON ESCAPE
precedence over ON KEY, 2-193
with READ, 2-194
with SET ESCAPE, 2-192
ON KEY
LABEL, 2-193
precedence, 2-193
with a GET, 2-193
with SET TYPEAHEAD, 2-192
with WAIT, 2-194
ON PAD, 2-197
ON PAGE, 1-21, 2-198
ON READERROR, 2-200
ON SELECTION PAD, 2-201
ON SELECTION POPUP
deactivating, 2-202
SAVE SCREEN, 2-202
OPEN (SQL), 6-9, 6-35, 6-43, 6-56, 6-57
Operator
defined, 1-12
in expression, 1-11
logical, 1-14
mathematical, 1-14
precedence, 1-15
relational, 1-14
string, 1-15
types of, 1-14
Options menu, 3-1
ORDER BY clause (SQL), 6-73, 6-74, 6-78
in UNION query, 6-73
use of index, 6-13
ORDER(), 4-95
OS(), 4-96
Output, streaming
assigning line number, 5-26
determining page number, 5-10
establishing line spacing, 5-33
page breaks, 2-136
positioning, 5-14
printer paper advance, 5-8
returning current line number, 5-26
Overwriting memos, 2-72

P

PACK, 2-204, 3-4
PAD message placement, 2-112.
See also Menu pads
PAD(), 4-97
_padvance, 5-8, 5-9

Page breaks in format files, 2-220
Page handler, 2-137, 2-198
Page handling from REPORT FORM, 2-198
_pageno, 5-10, 5-11
Page number, 5-10
Paragraph indent, 5-5
Parameter passing
to binary programs, 2-182
to UDFs, 2-144
with CALL, 2-53
with DO, 2-127
PARAMETERS, 1-10, 2-205
Parentheses
to group combined operations, 1-15
to group logical operations, 1-14
to group mathematical operations, 1-14
to group SQL search conditions, 6-69
to group string operations, 1-15
Password protection (SQL), 6-48
Path specification
CREATE DATABASE, 6-11
explicit, 1-5
saving, 1-5, 1-6
PAYMENT(), 4-98
_pbpage, 5-12, 5-13
PCOL(), 4-99
_pcolno, 5-14, 5-15
_pcopies, 5-16, 5-17
_pdriver, 5-18, 5-19
_pecode, 5-20
_peject, 5-21
_pepage, 5-22, 5-23
_pform, 5-24
PI(), 4-100
PICTURE functions
compared with _alignment, 5-1
PLAY MACRO, 2-206
_plength, 5-25
_plineno, 5-26
_ploffset, 5-27
Pop-up menu functions. *See* Menu functions
Pop-up menus, bar number, 4-13
Popup
deactivating, 2-99
menu, 2-114
option bars in, 2-107
window definition, 2-114
POPUP MESSAGE placement, 2-115
POPUP(), 4-101
PostScript printer, 5-19
_ppitch, 5-28, 5-29
_pquality, 5-30
Predicates (SQL), 6-70
.prf file, 1-5, 1-21
activating, 5-24
contents, 5-24
returning name of current, 5-24
.prg file
creating, 1-7
Print form
activating file, 5-24
returning name of current, 5-24
Print form file. *See* .prf file
Printer
drivers, F-1
pausing between pages, 5-34
returning current pitch, 5-28
returning print quality mode, 5-30
selecting quality print, 5-30
setting pitch, 5-28
Printing
boxes, 2-109
determining column, 4-99
determining row, 4-105
determining status, 4-102
from a program, 2-207
page number, control of, 2-207
UDF controls on, 1-19
Printing functions
PCOL(), 4-99
PRINTSTATUS(), 4-102
PROW(), 4-105
Printjob
activating driver, 5-18
ending control codes, 5-20
ending page eject, 5-21
paper advance, 5-8
preceding page eject, 5-21
setting number of copies, 5-16
setting output page length, 5-25
setting page left offset, 5-27
specifying beginning page, 5-12
specifying ending page, 5-22
supplying starting codes, 5-31
PRINTJOB/ENDPRINTOB, 2-207
PRINTSTATUS(), 4-102
PRIVATE, 1-10, 2-209
Privileges (SQL), 7-1, 7-4, 7-5
adding, 6-49
ALTER, 6-6, 6-48
assigning, 6-46
checked before FETCH, 6-43
cumulative, 6-49
DELETE, 6-48
granting granted, 6-49
INDEX, 6-48
INSERT, 6-48, 6-50, 6-53

removing, 6-58
required for SELECT, 6-65
SELECT, 6-48
UPDATE, 6-48
PRIVILEGES keyword (SQL), 6-48, 6-59
Procedure
in procedure file, 1-8
in program file, 1-8
limits, 1-9
list, 1-8
operation of, 1-7
search order, 1-9
PROCEDURE, 2-210
PROCEDURE keyword, 1-8
Procedure lists in .dbo files, 2-210
Procedures
determining name, 4-103
in a program file, 2-210
Production .mdx file, 2-155
Program
command line in, 1-1
creating, 1-7
defined, 1-7
determining name, 4-103
Program editor. *See* Text editor
Program file (SQL). *See* .prs file
Program file functions
CERROR, 4-18
ERROR(), 4-40
LINENO(), 4-72
MESSAGE(), 4-88
PROGRAM(), 4-103
Program file, dBASE. *See* .prg file
PROGRAM(), 4-103
Programming functions
CALL(), 4-15
CERROR(), 4-18
ERROR(), 4-40
INKEY(), 4-56
MESSAGE(), 4-88
PROGRAM(), 4-103
READKEY(), 4-108
PROMPT
for user input, 2-160
of DEFINE PAD, 2-112
of WAIT command, 2-279
option of DEFINE BAR, 2-107
option of DEFINE POPUP, 2-114
option of popups, 2-114
STRUCTURE of popups, 2-114
with DEFINE BAR, 2-114
Prompt messages, 4-104
PROMPT(), 4-104
PROTECT
in SQL, 6-26, 6-46, 6-48, 7-4
menu, 2-213
password in, 2-213
security levels, 2-213
PROW(), 4-105
.prs file (SQL)
creating, 1-7
_pscode, 5-31, 5-32
_pspacing, 5-33
PUBLIC, 1-10, 2-217
PUBLIC keyword (SQL), 6-48, 6-49, 6-58, 6-59
PV(), 4-106
_pwait, 5-34

Q

.qbe files, 1-5
.qbo file, 1-9
Query
clause, 1-5
file extensions, 2-86, 2-87
QUIT, 2-219
Quotes
in INSERT, 6-51
in SELECT search, 6-69
to delimit character constants, 1-12

R

Radians, 4-37, 4-119
RAND(), 4-107
RapidFile files
exporting to from SQL, 6-89
importing to SQL from, 6-54
READ, 2-220, 3-4
Reading in a file into a memo, 2-37
READKEY(), 4-108
RECALL, 2-222
RECCOUNT(), 4-110
RECNO(), 4-111
Record
buffer, 3-4
field name in, 1-11
RECORD keyword, 1-10
Record pointer
current, 4-111
determining position, 4-14, 4-39, 4-111
Records
deleted, 4-31
determining changes to, 4-20
determining number, 4-111

determining size, 4-112
locked, 4-73
locking, 4-115
selecting, 4-78, 4-122
RECSIZE(), 4-112
REINDEX, 2-223, 3-4, 6-28
Relational operators
listed, 1-14
precedence, 1-15
RELEASE, 2-224
RENAME, 2-226
Repeating characters, 4-113
REPLACE, 3-4, 2-228
with INDEX, 2-228
REPLACE FROM ARRAY, 2-230
REPLICATE(), 4-113
Report designer, 1-21, 2-88
REPORT FORM, 1-9, 1-21, 2-232
Report form files, 2-88
Reserved words (SQL), 6-2
RESET, 2-234
REST keyword, 1-10
RESTORE, 2-235
RESTORE MACROS, 2-236
RESTORE SCREENS, 2-237
RESTORE WINDOW, 2-238
RESUME, 2-239
RETRY, 2-240
RETURN, 1-17, 2-241
Returning from a UDF interrupt, 2-146
REVOKE (SQL), 6-47, 6-58, 6-59, 6-60, 7-4, 7-5
RIGHT(), 4-114
RLOCK(), 4-115
_rmargin, 5-35, 5-36
ROLLBACK, 2-242, 6-36, 6-61, 6-62
ROLLBACK(), 4-116
tests for, 2-42
returned, 2-42
ROUND(), 4-117
Rounding decimal places, 4-117
Routine. *See* Procedure
Row
for printing, 4-105
of cursor, 4-118
(SQL) cursor processing, 6-31, 6-56
(SQL) deleting, 6-34
(SQL) inserting, 6-50
(SQL) ordering, 6-73
(SQL) qualifying, 6-67, 6-68, 6-72
(SQL) summarizing information, 6-72
ROW(), 4-118
RTBLKSIZE, 2-174
RTMAXBLKS, 2-174
RTOD(), 4-119
RTRIM(), 4-120
RUN, 2-244
RUN(), 4-121
RUNSTATS (SQL), 6-28, 6-63, 6-64, 7-3, 7-6

S

Sample files, C-1
SAVE, 2-246
SAVE MACROS, 2-247
SAVE SCREEN, 2-248
SAVE TO TEMP clause (SQL), 6-74, 6-75
in UNION query, 6-73
SAVE WINDOW, 2-249
Saving keyboard macros to a file, 2-247
Saving memory variables to a file, 2-246
SCAN, 2-250
Scope, 1-10
Search condition (SQL)
defining, 6-67, 6-68
in HAVING, 6-72
SEEK
with SET DELETED, 2-252
with SET FILTER, 2-252
with SET NEAR, 2-252
SEEK(), 4-122
SELECT, 2-254
SELECT (SQL), 6-9, 6-65, 7-2
combining queries, 6-73
cursor definition, 6-31
cursor update, 6-32, 6-74
DECLARE CURSOR form, 6-80
defining query, 6-65
defining search condition, 6-67, 6-68
display-only form, 6-76
embedded single-row form, 6-79, 6-80
executing cursor, 6-56
grouping rows, 6-72
identifying tables, 6-67
in CREATE VIEW, 6-22
in HAVING, 6-72
in INSERT, 6-50
in UPDATE, 6-92
into memvars, 6-67
joining tables, 6-78
ordering result, 6-73
qualifying group rows, 6-72
qualifying rows, 6-67, 6-68
querying a view, 6-23
querying SYSIDXS, 6-14
querying SYSSYNS, 6-16

saving a result, 6-74, 6-75
SELECTing columns, 6-65, 6-66
subquery, 6-78
usage modes, 6-75
SELECT clause (SQL), 6-65, 6-66
SELECT privilege (SQL), 6-48, 6-88
SELECT(), 4-124
Self-join (SQL), 6-67
Semicolon (;)
as continuation character, 1-7
as SQL command terminator, 6-1
SET
disk, 3-1
display, 3-1
files, 3-1
in Config.db, 3-1
keys, 3-1
options, 3-1
SET ALTERNATE, 3-2
SET("ALTERNATE"), 4-125
SET AUTOSAVE, 3-4
in Config.db, 3-4
memo fields, 3-4
SET("AUTOSAVE"), 4-125
SET BELL, 3-5
default frequency, 3-5
duration, 3-5
frequency range, 3-5
in Config.db, 3-5
SET("BELL"), 4-125
SET BLOCKSIZE, 3-6
.dbt file, 3-6
default blocksize, 3-6
I/O processing, 3-6
in Config.db, 3-6
memo files, 3-6
SET("BLOCKSIZE"), 4-125
string manipulation, 3-6
SET BORDER, 3-7
allowed values, 3-7
and DEFINE WINDOW, 3-8
default border, 3-8
inverse video panel, 3-8
required order, 3-7
SET CARRY, 3-10
and indexes, 3-10
appending records, 3-10
blank records, 3-10
with INSERT, 3-10
SET("CARRY"), 4-125
SET CATALOG, 1-4, 3-11
adding entries, 3-14
alias, 3-13
and DOS PATH command, 3-12
and MODIFY REPORT, 3-12
and SET TITLE, 3-13
closing, 3-12
codes, 3-13
creating, 3-11
deleting files, 3-12
file title prompts, 3-14
file types, 3-13
filenames, 3-13
.frm files, 3-12
SET("CATALOG"), 4-125
updating, 3-11
with SET TITLE, 3-11
work areas, 3-11
SET CENTURY, 3-15
date format, 3-15
default date entry, 3-15
default display, 3-15
internal storage, 3-15
non-twentieth century dates, 3-15
SET("CENTURY"), 4-125
twentieth century dates, 3-15
SET CLOCK, 3-16
default setting, 3-16
full–screen menu commands, 3-16
SET("CLOCK"), 4-125
SET COLOR, 3-17
* attribute, 3-20
+ attribute, 3-20
and ACTIVATE WINDOW, 3-24
and DEFINE WINDOW, 3-24
and ISCOLOR(), 3-17
attribute characters, 3-21
background color, 3-18
background screen area, 3-17
COMPAQ computers, 3-24
display attributes, 3-17
enhanced area groups, 3-23
enhanced screen area, 3-17, 3-18
enhanced screen area components, 3-21
foreground color, 3-18
in Config.db, 3-17, 3-18
laptop computers, 3-24
modifying one area, 3-19
monochrome monitors, 3-19
perimeter screen area, 3-17
RGB colors, 3-19
SET("ATTRIBUTES"), 4-125
SET("COLOR"), 4-125
standard area groups, 3-21, 3-22
standard screen area, 3-17, 3-18
standard screen area components, 3-21
syntax, 3-19

SET commands
changing settings, 1-16
dBASE III PLUS, 1-16
determining status, 4-125
forms of, 1-16
full-screen menu, 1-16
notation, 1-2
usage, 1-2
used on command line, 1-1
SET CONFIRM, 3-26
cursor advancement, 3-26
SET("CONFIRM"), 4-125
SET CONSOLE, 3-27
dot prompt, 3-27
error messages, 3-27
output, 3-27
safety prompts, 3-27
SET("CONSOLE"), 4-125
SET CURRENCY, 3-28
separator character, 3-28
with FUNCTION clauses, 3-28
with PICTURE clauses, 3-28
SET CURRENCY LEFT/RIGHT, 3-29
default position, 3-29
in Config.db, 3-29
SET CURSOR, 3-30
and RUN command, 3-30
external programs, 3-30
SET("CURSOR"), 4-125
SET DATE, 1-13, 3-31
default format, 3-31
default setting, 3-31
in Config.db, 3-31
SET DBTRAP, 1-19, 3-32
active windows, 3-32
and DEACTIVATE WINDOW, 3-32
and DEFINE WINDOW, 3-32
and UDFs, 3-33
commands affected, 3-33
current environment, 3-32
database files, 3-32
windows, 3-32
work areas, 3-32
SET DEBUG, 3-34
and DISPLAY, 3-34
and LIST, 3-34
and SET ECHO, 3-34
and SET PRINTER, 3-34
and SET TALK, 3-34
rerouted output, 3-34
SET("DEBUG"), 4-125
SET DECIMALS, 3-35
calculations, 3-35
default setting, 3-35
financial calculations, 3-35
mathematical calculations, 3-35
numeric functions, 3-35
SET("DECIMALS"), 4-125
trigonometric calculations, 3-35
SET DEFAULT, 3-36
and SET DIRECTORY, 3-36
default drive, 3-36
drives and directories, 3-36
in Config.db, 3-36
SET("DEFAULT"), 4-125
SET DELETED, 3-37
and GO, 3-37
and INDEX, 3-37
and LIST, 3-37
and LOCATE, 3-37
and RECALL ALL, 3-37
and REINDEX, 3-37
and SET RELATION, 3-37
commands affected, 3-37
SET("DELETED"), 4-125
when effective, 3-37
SET DELIMITERS, 3-38
and @...GET, 3-39
and SET INTENSITY, 3-38
default character, 3-38
full–screen mode, 3-38
length of character string, 3-38
reset to default, 3-38
SET("DELIMITERS"), 4-125
SET DESIGN, 3-40
and CREATE VIEW FROM ENVIRONMENT, 3-40
and CREATE, 3-40
and MODIFY, 3-40
Control Center, 3-40
SET("DESIGN"), 4-125
SHIFT-F2 key, 3-40
SET DEVELOPMENT, 3-41
and DO, 3-41
Config.db, 3-41
external editors, 3-41
modify command, 3-41
object files, 3-41
program source files, 3-41
SET("DEVELOPMENT"), 4-125
TEDIT, 3-41
SET DEVICE, 3-42
and @...GET, 3-42
and @...SAY, 3-42
and EJECT, 3-42
and FILE keyword, 3-42
page eject, 3-42

SET DIRECTORY, 3-43
and SET DEFAULT, 3-43
in Config.db, 3-43
program completion, 3-43
SET("DIRECTORY"), 4-125
startup drive and directory, 3-43
SET DISPLAY, 3-45
external programs, 3-45
monitor types, 3-45
number of lines, 3-45
SET("DISPLAY"), 4-125
SET DOHISTORY, 1-16
SET ECHO, 3-46
and SET DEBUG, 3-46
and SET STEP, 3-46
and SET TALK, 3-46
debugging programs, 3-46
output, 3-46
SET("ECHO"), 4-125
SET ENCRYPTION, 3-47
access level, 3-47
and COPY, 3-47
and COPY STRUCTURE EXTENDED, 3-47
and COPY TO, 3-47
and EXPORT, 3-47
and JOIN, 3-47
and MODIFY STRUCTURE, 3-47
and PROTECT, 3-47
and TOTAL, 3-47
enciphered data, 3-47
group name, 3-47
password, 3-47
SET("ENCRYPTION"), 4-125
user name, 3-47
SET ESCAPE, 3-48
and CTRL-S, 3-48
and INKEY (), 3-48
and ON ESCAPE, 3-48
and ON KEY, 3-48
escape key, 3-48
program termination, 3-48
screen scrolling, 3-48
SET("ESCAPE"), 4-125
SET EXACT, 1-14, 3-49
program compilation, 3-49
SET("EXACT"), 4-125
string comparison, 3-49
SET EXCLUSIVE, 3-51
SET ("EXCLUSIVE"), 4-125
SET FIELDS, 3-52
adding fields, 3-53
aliases, 3-52, 3-54
and ALL EXCEPT, 3-52
and ALL LIKE, 3-52, 3-55
and COPY STRUCTURE, 3-53
and DISPLAY STRUCTURE, 3-53
and LIST STRUCTURE, 3-53
and LOCATE, 3-54
and SET CARRY, 3-53
and SET FILTER, 3-54
and SET RELATION, 3-53, 3-54
commands affected, 3-53
data integrity, 3-54
default fields, 3-52
expression list, 3-52
field lists, 3-52
field options, 3-52
SET("FIELDS"), 4-125
SET FILTER, 3-56
activating, 3-56
and SET CATALOG, 3-56
commands affected, 3-56
field types, 3-56
queries, 3-56
SET("FILTER"), 4-125
work areas, 3-56
SET FIXED, 1-16
SET FORMAT, 3-58
@ commands, 3-58
and @...GET, 3-59
and READ, 3-58
catalogs, 3-59
compiled files, 3-58
default entry form, 3-59
memory variables, 3-58
MODIFY SCREEN, 3-58
screen files, 3-58
SET("FORMAT"), 4-125
standard display, 3-59
work areas, 3-59
SET FULLPATH, 3-60
dot prompt, 3-60
in Config.db, 3-60
SET("FULLPATH"), 4-125
SET FUNCTION, 3-61
and ON KEY, 3-61, 3-62
default function key assignments, 3-61
Enhanced keyboard, 3-61
Help key, 3-61
in Config.db, 3-61
Maximum characters, 3-61
programming keys, 3-62
SET HEADINGS, 3-63
column titles, 3-63
column width, 3-63
SET("HEADINGS"), 4-125

SET HELP, 3-64
SET("HELP"), 4-125
SET HISTORY, 1-3, 3-65
memory resident, 3-65
memory usage, 3-65
program commands, 3-65
SET("HISTORY"), 4-125
SET HOURS, 3-66
in Config.db, 3-66
SET("HOURS"), 4-125
system clock, 3-66
SET INDEX, 3-67
and APPEND, 3-68
and FIND, 3-67
and INSERT, 3-68
and SEEK, 3-67
and SET ORDER, 3-67
natural order, 3-68
order clause, 3-67
updating, 3-68
work areas, 3-67
SET INSTRUCT, 3-70
and CREATE/MODIFY, 3-70
and SET TALK, 3-70
SET("INSTRUCT"), 4-125
SET INTENSITY, 3-71
and COLOR OF FIELDS, 3-71
and inverse video, 3-71
and SET COLOR OF NORMAL, 3-71
and SET DELIMITERS, 3-71
SET("INTENSITY"), 4-125
SET LOCK, 3-72
and networks, 3-72
and REPORT FORM, 3-72
automatic locking, 3-72
commands affected, 3-72
data integrity, 3-72
lock status, 3-72
read-only status, 3-72
SET("LOCK"), 4-126
SET MARGIN, 3-73
and LIST STATUS, 3-74
in Config.db, 3-73
printer offset, 3-73
printer output, 3-73
SET("MARGIN"), 4-126
SET MARK, 3-75
delimiter character, 3-75
entry format, 3-75
SET MEMOWIDTH, 3-76
?/?? commands, 3-76
default width, 3-76
maximum width, 3-76
SET("MEMOWIDTH"), 4-126
vertical stretch, 3-76
SET MENU, 1-16
SET MESSAGE, 3-77
and SET SCOREBOARD, 3-77
maximum characters, 3-77
message line, 3-77
positioning, 3-77
SET NEAR, 3-78
and SET DELETED, 3-78
and SET FILTER, 3-78
exact matches, 3-78
record pointer, 3-78
SET("NEAR"), 4-126
SET ODOMETER, 3-80
SET("ODOMETER"), 4-126
SET ORDER, 3-81
and LOCATE, 3-81
and SET ORDER TO TAG, 3-81
by filename, 3-81
reassigning files, 3-81
SET PATH, 1-6, 3-83
and CREATE, 3-83
and DOS PATH command, 3-83
and SET DEFAULT, 3-83
and SET DIRECTORY, 3-83
maximum length, 3-83
SET("PATH"), 4-126
SET PAUSE, 3-84, 6-67
and DISPLAY, 3-84
and SQL SELECT, 3-84
column headings, 3-84
SET("PAUSE"), 4-126
SET POINT, 3-85
decimal separator, 3-85
printed output, 3-85
screen display, 3-85
SET PRECISION, 3-86
in Config.db, 3-86
numeric accuracy, 3-86
SET("PRECISION"), 4-126
SET PRINTER, 3-87
and @...SAY, 3-87
and CAPTURE, 3-87
and SPOOLER, 3-87
embedding codes, 3-89
HP Laserjet printers, 3-89
local devices, 3-88
network printing, 3-88
printing device, 3-87
screen output, 3-87
SET("PRINTER"), 4-126
to DOS files, 3-89

SET PROCEDURE, 1-8, 1-9, 3-91
file extensions, 3-91
maximum number of procedures, 3-91
maximum RAM, 3-91
SET("PROCEDURE"), 4-126
SET REFRESH, 3-93
in Config.db, 3-93
in LAN environment, 3-93
locked fields or files, 3-93
records, read, 3-93
SET("REFRESH"), 4-126
SET RELATION, 3,95
and CREATE VIEW, 3-96
and SET NEAR, 3-95
and SET SKIP, 3-95
child files, 3-95
circular relations, 3-95
parent files, 3-95
relating expressions, 3-95
SET("RELATION"), 4-126
work areas, 3-95
SET REPROCESS, 3-99
escaping retry, 3-99
number of retries, 3-99
SET("REPROCESS"), 4-126
SET SAFETY, 3-100
and MODIFY, 3-100
and ZAP, 3-100
overwriting data, 3-100
SET("SAFETY"), 4-126
SET SCOREBOARD, 3-101
and SET STATUS, 3-101
scoreboard indicators, 3-101
SET("SCOREBOARD"), 4-126
SET SEPARATOR, 3-102
in Config.db, 3-102
thousands separator, 3-102
SET SKIP, 3-103
and LIST, 3-103
and SET RELATION, 3-103
records processed, 3-103
work areas, 3-103
SET SPACE, 3-105
?/?? commands, 3-105
dBASE III PLUS, 3-105
SET("SPACE"), 4-126
SET SQL, 1-3, 3-106
at dot prompt, 3-106
in Config.db, 3-106
in programs, 3-106
SET("SQL"), 4-126
SET STATUS,, 3-107
commands affected, 3-107
in Config.db, 3-107
information displayed, 3-107
lowest usable line, 3-107
SET("STATUS"), 4-126
SET STEP, 3-108
debugging programs, 3-108
SET("STEP"), 4-126
step options, 3-108
SET TALK, 3-109
commands affected, 3-109
compiler warning messages, 3-109
SET("TALK"), 4-126
SET TITLE, 3-110
catalogs, 3-110
or description, 3-110
SET("TITLE"), 4-126
work areas, 3-110
SET TRAP, 3-111
debugger, 3-111
error options, 3-111
escape key, 3-111
SET("TRAP"), 4-126
SET TYPEAHEAD, 3-112
and ON ERROR, 3-112
and ON KEY, 3-112
and SET BELL, 3-112
maximum buffer, 3-112
SET("TYPEAHEAD"), 4-126
SET UNIQUE, 3-113
and INDEX ON, 3-113
and REINDEX, 3-113
MDX files, 3-113
NDX files, 3-113
SET("UNIQUE"), 4-126
SET VIEW, 3-114
and CREATE/MODIFY, 3-115
and SET CATALOG, 3-115
and SET TITLE, 3-115
compiled files, 3-115
contents of views, 3-115
queries, 3-115
SET("VIEW"), 4-126
SET WINDOW, 3-117
memo fields, 3-117
previously defined windows, 3-117
SET(), 4-125
SHOW DATABASE (SQL), 6-84
SHOW MENU, 2-256
SHOW POPUP, 2-257
SIGN(), 4-128
Simple subquery (SQL), 6-78
SIN(), 4-129
Sine, 4-129
Single index, 6-28
Single-database application (SQL), 6-86

Skeleton, 1-11
SKIP, 2-258
SORT, 2-260
SOUNDEX(), 4-130
Source code, 1-7
SPACE(), 4-132
Specifications, B-1
SQL mode, 1-3
SQL program in UDFs, 1-19
SQL prompt, 1-3
Sqlcnt status variable (SQL), 6-44, 6-56
Sqlcode status variable (SQL), 6-44, 6-56
SQLDATABASE (SQL). *See* Config.db file
SQLDBA user ID (SQL)
  ALL privileges, 6-48
  catalog privileges, 7-2
  DROP privilege, 6-12, 6-14, 6-16, 6-22, 6-38, 6-39, 6-40, 6-41, 6-42
  REVOKE privilege, 6-59
SQLHOME command (SQL). *See* Config.db file
SQLHOME directory (SQL), 7-1
SQRT(), 1-13, 4-133
Square root, 4-133
START DATABASE (SQL), 1-3, 6-9, 6-85
Statistics (SQL), 6-63, 7-3
STOP DATABASE (SQL), 6-9, 6-38, 6-85, 6-87
STORE, 1-10, 2-263
Storing averages to an array, 2-40
STR(), 4-134
String operators
  listed, 1-15
  precedence, 1-15
Structure extended files
  backward compatibility, 2-75
  index tags, 2-75
STUFF(), 4-135
Subquery (SQL)
  correlated, 6-78
  simple, 6-78
Subroutine. *See* PROCEDURE
SUBSTR(), 4-136
Substrings
  extracting, 4-136
  LEFT(), 4-69
  position of, 4-10
  RIGHT(), 4-114
SUM, 1-10, 2-266
SUSPEND, 2-267
Synonym (SQL)
  catalog entry, 7-1, 7-8
  creating, 6-16
  deleting, 6-40
  effect of DROP TABLE, 6-41
  effect of DROP VIEW, 6-42
  in INSERT, 6-50
  in SELECT clause, 6-65, 6-67
  naming, 6-16
  use, 6-16
Syntax
  clauses in, 1-9
  command verb in, 1-9
  general form, 1-10
  notation, 1-10
Syntax (SQL), 6-1
SYSAUTH catalog (SQL), 7-1, 7-4
SYSCOLAU catalog (SQL), 7-1, 7-5
SYSCOLS catalog (SQL), 7-1, 7-5
SYSDBS catalog (SQL), 6-11, 6-84, 7-1, 7-6
SYSIDXS catalog (SQL), 6-14, 7-1, 7-7
SYSKEYS catalog (SQL), 7-1, 7-7
SYSSYNS catalog (SQL), 6-16, 7-1, 7-8
SYSTABLS catalog (SQL), 7-1, 7-8
System Data Format files
  exporting to from SQL, 6-89
  importing to SQL from, 6-54
System date, 4-28
System parameters
  changing, 1-16
  COL(), 4-24
  DISKSPACE(), 4-33
  GETENV(), 4-53
  ISCOLOR(), 4-63
  MEMORY(), 4-86
  NETWORK(), 4-94
  OS(), 4-96
  PCOL(), 4-99
  PRINTSTATUS(), 4-102
  PROW(), 4-105
  ROW(), 4-118
  SET(), 4-125
  TIME(), 4-139
  VERSION(), 4-149
SYSTIMES catalog (SQL), 7-2, 7-9
SYSVDEPS catalog (SQL), 7-2, 7-9
SYSVIEWS catalog (SQL), 6-21, 7-2, 7-9

T

Table (SQL)
  adding columns, 6-6
  alias name for, 6-67
  appending data, 6-53
  assigning privileges to, 6-46
  catalog entry, 7-1, 7-8
  creating, 6-18

deleting, 6-41
exporting data, 6-88
inserting rows, 6-50
naming, 6-18
querying, 6-65
restrictions on, 6-20
SELECTing from, 6-67
synonym for, 6-16
updating statistics about, 6-63
TABLE keyword (SQL), 6-48, 6-59
Tabs, 5-37
_tabs, 5-37
TAG(), 4-137
TAN(), 4-138
TEXT, 2-268
Text editor
writing programs, 1-7
using in window, 1-3
TIME(), 4-139
TO keyword, 1-10
TOTAL, 2-269
Transaction (SQL)
limited commands, 2-43
removing SQL changes, 6-61
RESET, 2-44
restricted commands, 2-42
ROLLBACK, 2-42
SQL commands disallowed in, 6-61
TRANSFORM(), 4-140
Trapping Esc key
after a READ, 2-194
with ON KEY, 2-193
Trapping keys, 2-193
Trigonometric functions
ACOS(), 4-6
ASIN(), 4-9
ATAN(), 4-11
ATN2(), 4-12
COS(), 4-26
DTOR(), 4-37
PI(), 4-100
RTOD(), 4-119
SIN(), 4-129
TAN(), 4-138
TRIM(), 4-120, 4-141
TYPE, 2-271
Type F numbers, 1-13
Type N numbers, 1-13
TYPE(), 4-142
Type, of expression, 4-142

U

UDF
blocked commands, 2-145
creating, 1-17
illustrated, 1-18
indexing, 2-147
intervening commands, 1-19
limitations, 1-18
macros, 2-147
names, 2-144, 2-145
non-recursive commands, 2-145
permitted changes, 2-145
rules, 1-19
SET DBTRAP, 3-32
syntax, 1-17
Undeleting records. *See* RECALL
UNION clause (SQL), 6-73, 6-76
UNLOAD DATA (SQL), 6-29, 6-55, 6-88, 6-89, 6-90
UNLOCK, 2-272
.upd files, 1-5
Updatable view (SQL)
creating, 6-23, 6-24
restrictions, 6-23, 6-24
UPDATE, 2-273
UPDATE (SQL), 6-63, 6-74, 6-78, 6-91, 6-92, 6-93
embedded form, 6-80
WHERE clause, 6-92
WHERE CURRENT OF clause, 6-74, 6-92, 6-93
UPDATE privilege (SQL), 6-48, 6-92
UPPER(), 4-144
Uppercase
checking, 4-66
conversion, 4-144
USE, 2-275
ALIAS option, 1-6
IN clause, 1-11
User
access level, 4-4
name of, 4-73, 4-145
User ID (SQL)
assigning privileges to, 6-48
returning, 6-66
User input
into memory variables, 2-220
memory variables, 2-160
prompt, 2-160
USER keyword (SQL), 6-66
USER(), 4-145
User-defined function. *See* UDF

V

VAL(), 4-146
VARREAD(), 4-147
VERSION(), 4-149
View (SQL)
  alias name for, 6-67
  assigning privileges to, 6-46
  catalog entry, 7-1, 7-2, 7-8, 7-9
  creating, 6-21
  deleting, 6-42
  effect of DROP TABLE, 6-41
  inserting via, 6-50
  naming, 6-21
  naming columns, 6-22
  querying, 6-22, 6-23, 6-65
  SELECT use, 6-21
  SELECTing from, 6-65
  synonym for, 6-16
  updatable, 6-23, 6-24, 6-50
View, dBASE
  effect of DBDEFINE, 6-28
Virtual table. *See* View (SQL)
VisiCalc files
  exporting to from SQL, 6-89
  importing to SQL from, 6-54
.vue files, 1-5

W

WAIT, 1-10
  capturing the keypress, 2-279
  with SET ESCAPE, 2-279
WHERE clause (SQL)
  in DELETE, 6-35
  in UPDATE, 6-92
  qualifying rows, 6-67, 6-68
  specifying a join, 6-78
WHERE CURRENT OF clause (SQL)
  in DELETE, 6-32, 6-34, 6-35, 6-36
WHILE clause, 1-10
Wildcards, in comparisons, 4-71
Window
  border defaults, 2-116
  COLOR option, 2-116
WINDOW(), 4-150
WITH CHECK OPTION (SQL), 6-24
WITH GRANT OPTION (SQL), 6-49, 6-59
Word wrap
  setting default tabs, 5-37
  setting on/off, 5-38
Work area
  alias name, 4-7
  letter identifier, 1-6
  number identifier, 1-6
  SELECT(), 4-124
  using IN to select, 1-11
WORK keyword (SQL), 6-61
_wrap, 5-38

Y

YEAR(), 4-151

Z

ZAP, 2-280, 3-4

NOTES