

# *dBASE IV*<sup>®</sup>

Version 2.0

---

## Language Reference

BORLAND INTERNATIONAL, INC. 1800 GREEN HILLS ROAD  
P.O. BOX 660001, SCOTTS VALLEY, CA 95067-0001

Copyright © 1984, 1993 by Borland International. All Rights Reserved. All Borland products are trademarks or registered trademarks of Borland International, Inc. Other brand and product names are trademarks or registered trademarks of their respective holders.

**PRINTED IN IRELAND**

10 9 8 7 6 5 4 3 2 1

# Contents

<b>Introduction</b> .....	1
Chapter 1: Essentials .....	1
Chapter 2: Commands .....	1
Chapter 3: SET Commands .....	1
Chapter 4: Functions .....	2
Chapter 5: System Memory Variables .....	2
Chapter 6: SQL Commands .....	2
Chapter 7: SQL Catalogs .....	2
Appendixes .....	2

## **Essentials**..... 3

<b>Chapter 1, Essentials</b> .....	5
About This Chapter .....	5
dBASE IV Language Components .....	5
Dot Prompt Interface .....	6
SQL Prompt Interface .....	7
Filenames and Aliases .....	8
Programs and Procedures .....	11
Using Commands .....	13
Syntax .....	13
Expressions .....	15
Data Types .....	16
Operators .....	18
Precedence of Operators .....	19
Using SET Commands .....	20
Using Functions .....	20
User-Defined Functions .....	21
What is a UDF? .....	21
Example of a UDF .....	22
Limitations on UDFs .....	23
Using System Memory Variables .....	24

<b>Commands</b> .....	27
<b>Chapter 2, Commands</b> .....	29
???	29
???	33
@ .....	37
Format Functions .....	41
@...CLEAR .....	46
@...FILL .....	47
@...SCROLL .....	48
@...TO .....	49
ACCEPT .....	51
ACTIVATE MENU .....	52
ACTIVATE POPUP .....	53
ACTIVATE SCREEN .....	53
ACTIVATE WINDOW .....	54
APPEND .....	55
APPEND FROM .....	56
APPEND FROM ARRAY .....	59
APPEND MEMO .....	61
ASSIST .....	62
AVERAGE .....	63
BEGIN/END TRANSACTION .....	64
BLANK .....	68
BROWSE .....	70
CALCULATE .....	75
CALL .....	77
CANCEL .....	78
CHANGE .....	79
CLEAR .....	80
CLOSE .....	81
COMPILE .....	82
CONTINUE .....	86
CONVERT .....	87
COPY .....	88
COPY FILE .....	91
COPY INDEXES .....	92
COPY MEMO .....	93
COPY STRUCTURE .....	94
COPY STRUCTURE EXTENDED .....	95
COPY TAG .....	96
COPY TO ARRAY .....	97
COUNT .....	99
CREATE or MODIFY STRUCTURE .....	100

CREATE FROM .....	103
CREATE/MODIFY APPLICATION .....	104
CREATE/MODIFY LABEL .....	106
CREATE/MODIFY QUERY VIEW .....	107
CREATE/MODIFY REPORT .....	109
CREATE/MODIFY SCREEN .....	111
CREATE VIEW FROM ENVIRONMENT .....	113
DEACTIVATE MENU .....	114
DEACTIVATE POPUP .....	115
DEACTIVATE WINDOW .....	116
DEBUG .....	117
DECLARE .....	119
DEFINE BAR .....	122
DEFINE BOX .....	123
DEFINE MENU .....	125
DEFINE PAD .....	127
DEFINE POPUP .....	129
DEFINE WINDOW .....	130
DELETE .....	132
DELETE FILE .....	133
DELETE TAG .....	133
DEXPORT .....	134
DIR .....	135
DISPLAY .....	136
DO .....	137
DO CASE/ENDCASE .....	141
DO WHILE/ENDDO .....	142
EDIT .....	144
EJECT .....	147
EJECT PAGE .....	147
ERASE .....	148
EXPORT .....	149
FIND .....	150
FUNCTION .....	153
GO/GOTO .....	157
HELP .....	159
IF/ENDIF .....	160
IMPORT .....	161
INDEX .....	162
INPUT .....	168
INSERT .....	169
JOIN .....	170
KEYBOARD .....	172
LABEL FORM .....	174

LIST/DISPLAY .....	175
LIST/DISPLAY FILES .....	177
LIST/DISPLAY HISTORY .....	178
LIST/DISPLAY MEMORY .....	178
LIST/DISPLAY STATUS .....	181
LIST/DISPLAY STRUCTURE .....	182
LIST/DISPLAY USERS .....	184
LOAD .....	184
LOCATE .....	188
LOGOUT .....	190
MODIFY COMMAND/FILE .....	190
MOVE WINDOW .....	192
NOTE .....	193
ON BAR .....	193
ON ERROR/ESCAPE/KEY .....	195
ON EXIT BAR .....	199
ON EXIT MENU .....	201
ON EXIT PAD .....	203
ON EXIT POPUP .....	204
ON MENU .....	206
ON MOUSE .....	207
ON PAD .....	208
ON PAGE .....	209
ON POPUP .....	212
ON READERROR .....	213
ON SELECTION BAR .....	214
ON SELECTION MENU .....	215
ON SELECTION PAD .....	216
ON SELECTION POPUP .....	217
PACK .....	219
PARAMETERS .....	219
PLAY MACRO .....	220
PRINTJOB/ENDPRINTJOB .....	222
PRIVATE .....	224
PROCEDURE .....	224
PROTECT .....	227
PUBLIC .....	231
QUIT .....	232
READ .....	233
RECALL .....	234
REINDEX .....	235
RELEASE .....	236
RENAME .....	238
REPLACE .....	239

REPLACE FROM ARRAY .....	241
REPORT FORM .....	242
RESET .....	244
RESTORE .....	245
RESTORE MACROS .....	246
RESTORE SCREEN .....	247
RESTORE WINDOW .....	247
RESUME .....	248
RETRY .....	248
RETURN .....	249
ROLLBACK .....	250
RUN .....	251
SAVE .....	252
SAVE MACROS .....	253
SAVE SCREEN .....	254
SAVE WINDOW .....	255
SCAN/ENDSCAN .....	256
SEEK .....	257
SELECT .....	258
SHOW MENU .....	260
SHOW POPUP .....	260
SKIP .....	261
SORT .....	262
STORE .....	265
SUM .....	267
SUSPEND .....	268
TEXT/ENDTEXT .....	269
TOTAL .....	270
TYPE .....	272
UNLOCK .....	273
UPDATE .....	274
USE .....	275
WAIT .....	279
ZAP .....	280

## **SET Commands .....** 281

<b>Chapter 3, SET Commands .....</b>	<b>283</b>
SET .....	283
SET ALTERNATE .....	284
SET AUTOSAVE .....	286
SET BELL .....	286
SET BLOCKSIZE .....	287

SET BORDER .....	288
SET CARRY .....	290
SET CATALOG .....	291
SET CENTURY .....	294
SET CLOCK .....	295
SET COLOR .....	296
SET CONFIRM .....	304
SET CONSOLE .....	305
SET CURRENCY .....	305
SET CURRENCY LEFT/RIGHT .....	306
SET CURSOR .....	307
SET DATE .....	308
SET DBTRAP .....	309
SET DEBUG .....	311
SET DECIMALS .....	311
SET DEFAULT .....	312
SET DELETED .....	313
SET DELIMITERS .....	314
SET DESIGN .....	316
SET DEVELOPMENT .....	316
SET DEVICE .....	317
SET DIRECTORY .....	318
SET DISPLAY .....	319
SET ECHO .....	320
SET ENCRYPTION .....	321
SET ESCAPE .....	322
SET EXACT .....	322
SET EXCLUSIVE .....	324
SET FIELDS .....	324
SET FILTER .....	328
SET FORMAT .....	329
SET FULLPATH .....	331
SET FUNCTION .....	332
SET HEADINGS .....	333
SET HELP .....	334
SET HISTORY .....	335
SET HOURS .....	336
SET IBLOCK .....	337
SET INDEX .....	339
SET INSTRUCT .....	341
SET INTENSITY .....	342
SET KEY .....	343
SET LDCHECK .....	344
SET LIBRARY .....	345



SET LOCK .....	346
SET MARGIN .....	349
SET MARK .....	350
SET MBLOCK .....	351
SET MEMOWIDTH .....	352
SET MESSAGE .....	353
SET MOUSE .....	354
SET NEAR .....	354
SET ODOMETER .....	356
SET ORDER .....	356
SET PATH .....	358
SET PAUSE .....	359
SET POINT .....	360
SET PRECISION .....	360
SET PRINTER .....	361
SET PROCEDURE .....	364
SET REFRESH .....	366
SET RELATION .....	367
SET REPROCESS .....	371
SET SAFETY .....	371
SET SCOREBOARD .....	372
SET SEPARATOR .....	373
SET SKIP .....	374
SET SPACE .....	376
SET SQL .....	376
SET STATUS .....	377
SET STEP .....	378
SET TALK .....	378
SET TITLE .....	379
SET TRAP .....	380
SET TYPEAHEAD .....	381
SET UNIQUE .....	381
SET VIEW .....	383
SET WINDOW .....	384

## **Functions** .....

<b>Chapter 4, Functions</b> .....	389
<b>&amp;</b> .....	389
<b>ABS()</b> .....	390
<b>ACCESS()</b> .....	391
<b>ACOS()</b> .....	393
<b>ALIAS()</b> .....	394

ASC()	394
ASIN()	395
AT()	396
ATAN()	397
ATN2()	397
BAR()	398
BARCOUNT()	399
BARPROMPT()	400
BOF()	401
CALL()	403
CATALOG()	404
CDOW()	405
CEILING()	405
CERROR()	406
CHANGE()	407
CHR()	408
CMONTH()	410
COL()	411
COMPLETED()	412
COS()	413
CTOD()	413
DATE()	414
DAY()	415
DBF()	416
DELETED()	417
DESCENDING()	418
DGEN()	419
DIFFERENCE()	420
DISKSPACE()	421
DMY()	421
DOW()	422
DTOC()	423
DTOR()	424
DTOS()	425
EOF()	425
ERROR()	426
EXP()	427
FCLOSE()	428
FCREATE()	429
FDATE()	430
FEOF()	431
FERROR()	432
FFLUSH()	433
FGETS()	434

FIELD()	435
FILE()	436
FIXED()	437
FKLABEL()	438
FKMAX()	438
FLDCOUNT()	439
FLDLIST()	440
FLOAT()	441
FLOCK()	442
FLOOR()	443
FOPEN()	444
FOR()	445
FOUND()	447
FPUTS()	448
FREAD()	449
FSEEK()	451
FSIZE()	452
FTIME()	452
FV()	453
FWRITE()	454
GETENV()	455
HOME()	456
ID()	457
IIF()	457
INKEY()	459
INT()	463
ISALPHA()	464
ISBLANK()	464
ISCOLOR()	466
ISLOWER()	467
ISMARKED()	467
ISMOUSE()	468
ISUPPER()	469
KEY()	469
KEYMATCH()	470
LASTKEY()	472
LEFT()	473
LEN()	474
LIKE()	475
LINENO()	476
LKSYS()	477
LOCK()	479
LOG()	479
LOG10()	480

LOOKUP()	481
LOWER()	482
LTRIM()	483
LUPDATE()	483
MAX()	484
MCOL()	485
MDX()	486
MDY()	487
MEMLINES()	488
MEMORY()	489
MENU()	490
MESSAGE()	491
MIN()	491
MLINE()	492
MOD()	493
MONTH()	494
MROW()	495
NDX()	496
NETWORK()	497
ORDER()	498
OS()	499
PAD()	499
PADPROMPT()	500
PAYMENT()	501
PCOL()	502
PCOUNT()	502
PI()	503
POPUP()	504
PRINTSTATUS()	505
PROGRAM()	506
PROMPT()	506
PROW()	507
PV()	508
RAND()	509
RAT()	510
READKEY()	511
RECCOUNT()	512
RECNO()	513
RECSIZE()	514
REPLICATE()	515
RIGHT()	516
RLOCK()	516
ROLLBACK()	517
ROUND()	518

ROW()	519
RTOD()	519
RTRIM()	520
RUN()	521
SEEK()	523
SELECT()	525
SET()	526
SIGN()	530
SIN()	531
SOUNDEX()	531
SPACE()	533
SQRT()	533
STR()	534
STUFF()	535
SUBSTR()	536
TAG()	537
TAGCOUNT()	537
TAGNO()	538
TAN()	540
TIME()	541
TRANSFORM()	541
TRIM()	542
TYPE()	543
UNIQUE()	544
UPPER()	545
USER()	547
VAL()	547
VARREAD()	548
VERSION()	549
WINDOW()	550
YEAR()	550

## **System Memory Variables** ..... 553

<b>Chapter 5, System Memory Variables</b> .....	555
_alignment .....	555
_box .....	556
_indent .....	557
_lmargin .....	558
_padvance .....	559
_pageno .....	561
_pbpage .....	562
_pcolno .....	564

_pcopies	565
_pdriver	566
_pecode	568
_peject	569
_pepage	570
_pform	571
_plength	572
_plineno	573
_poffset	574
_ppitch	575
_pquality	576
_pscode	577
_pspacing	578
_pwait	579
_rmargin	580
_tabs	582
_wrap	583

## **SQL Commands** ..... 585

<b>Chapter 6, SQL Commands</b>	587
Symbols and Conventions	587
Reserved Words	588
Classes of Commands	589
Creation/Start-up of SQL Databases	589
Creation/Modification of Objects	589
Database Security	589
Data Definition	590
Deletion of Objects	590
Embedded SQL	591
Query and Update of Data	591
Utility	591
ALTER TABLE	592
CLOSE	594
CREATE DATABASE	596
CREATE INDEX	598
CREATE SYNONYM	600
CREATE TABLE	602
CREATE VIEW	605
DBCHECK	609
DBDEFINE	610
DECLARE CURSOR	612
DELETE	615

DROP DATABASE .....	618
DROP INDEX .....	619
DROP SYNONYM .....	620
DROP TABLE .....	620
DROP VIEW .....	621
FETCH .....	622
GRANT .....	624
INSERT .....	627
LOAD DATA .....	630
OPEN .....	633
REVOKE .....	634
ROLLBACK .....	637
RUNSTATS .....	638
SELECT .....	639
SHOW DATABASE .....	656
START DATABASE .....	657
STOP DATABASE .....	658
UNLOAD DATA .....	659
UPDATE .....	661

## **SQL Catalogs** .....

<b>Chapter 7, SQL Catalogs</b> .....	667
Updating the Catalog Tables .....	668
Description of Catalog Tables .....	670
Sysauth Table .....	670
Syscolau Table .....	671
Syscols Table .....	671
Sysdbs Table .....	672
Sysidxs Table .....	673
Syskeys .....	673
Sysstats Table .....	674
Systabls Table .....	674
Systimes Table .....	675
Sysvdeps Table .....	675
Sysviews Table .....	675

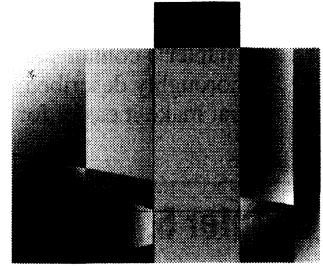
## **Appendixes** .....

<b>Appendix A, dBASE IV Specifications</b> .....	679
Database File .....	679
Index File .....	679

Field Sizes .....	679
Arrays .....	679
Multi-User Procedures .....	680
File Operations .....	680
Numeric Accuracy .....	680
Memory Variables .....	680
Compile-Time Symbols .....	681
Capacities .....	681
Word Wrap Editor .....	681
Forms .....	681
Reports .....	681
Labels .....	681
QBE .....	682
SQL .....	682
Applications Generator .....	682
Miscellaneous Capacities .....	682
<b>Appendix B, Sample Files .....</b>	<b>683</b>
Client.dbf .....	683
Transact.dbf .....	685
Stock.dbf .....	687
Menus.prg .....	688
<b>Appendix C, File Extensions .....</b>	<b>691</b>
File Extensions Used by dBASE IV .....	691
File Relations and Compatibilities .....	694
<b>Appendix D, Structure of a Database (.dbf) File .....</b>	<b>695</b>
Database Header and Records .....	695
Database Header Structure .....	695
Database Records .....	696
Memo Fields and the .dbt File .....	697
<b>Appendix E, ASCII Chart .....</b>	<b>699</b>
<b>Appendix F, dBASE Commands and Functions Allowed in SQL Mode .....</b>	<b>705</b>
dBASE Commands Allowed in SQL Mode .....	705
dBASE Functions Allowed in SQL Mode .....	707
<b>Appendix G, dBASE Error Messages .....</b>	<b>709</b>
Unrecoverable Errors .....	709
Error Messages .....	709
<b>Index .....</b>	<b>771</b>



# Introduction



The *Language Reference* is an encyclopedia of dBASE IV® commands, functions, and system memory variables. This manual is for users who have completed *Getting Started* or *Using dBASE IV*. If you are already familiar with dBASE® programming or are using the dBASE Compiler for DOS, you will find this manual a useful reference tool.

---

## Chapter 1: Essentials

This is a discussion of the dBASE IV basics: using commands, SET commands, functions, and system memory variables. It covers dBASE IV language components and shows you 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 dBASE IV. 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 B, "Sample Files." Also in Appendix B 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 an alphabetical list of the dBASE IV functions. Each function is thoroughly described, and 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, descriptions, and examples for each SQL command.

---

## Chapter 7: SQL Catalogs

This chapter describes SQL catalog tables.

---

## Appendixes

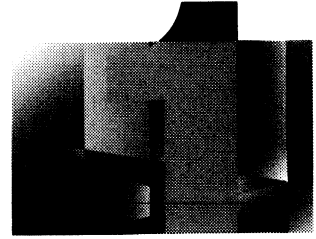
The appendixes cover the following topics:

- 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
- The ASCII chart, which provides hexadecimal and decimal equivalents for each characters
- dBASE IV commands and functions that can be used with SQL
- dBASE IV error messages, their numbers, and appropriate corrective actions

# Essentials



# Essentials



---

## 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 DATE sets the default date format.
- 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 the dBASE IV 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.

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.

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

### Filenames

A filename may be the actual name of a file as it is written on disk, or an *indirect reference* to the filename. Indirect file references are discussed later in this section. You may also use a drive specifier and path name along with the filename 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 C provides a complete listing of the dBASE IV file extensions.

 **NOTE** Please read the section on aliases for limitations with valid DOS filenames.

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 in the current directory and whether you provided a 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 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.

## Aliases

Each session of dBASE IV allows up to 40 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. For work areas greater than ten specify the work area by number or alias.

Work area numbers are from 1 to 40. 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 the work area number prefaced with an underscore (\_) 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 an SQL program.

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

In SQL, the semicolon marks the end of the command and is not read as the continuation mark.

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 versions of the dBASE product, 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 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.

Procedures may be grouped into system level procedures, program files, procedure files, and library files. Please see *Programming in dBASE IV* for a complete explanation regarding application and coding design issues.

System level procedures are a file of user-defined procedures and functions to augment the built-in commands and functions of dBASE IV. They are specified in Config.db with SYSPROC = <filename>.

This system procedure file will remain active for the life of a dBASE session.

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

1. Look for the procedure in SYSPROC = <filename>, if one is active.
2. Search the currently executing object (.dbo) file.
3. Search the SET PROCEDURE file, if one is active, for the first procedure with that name.
4. Search other open object files (most recently opened first).
5. Search the SET LIBRARY file, if one is active. This file is a collection of procedures and functions that augment the dBASE IV commands.
6. Find and open an object (.dbo) file of that name.
7. Find and compile a program (.prg) file of that name.
8. Find and compile a SQL program (.prs) file of that name.

This search order allows you to hide procedures with the same name from one another.

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 and COMPILE, 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 unless it is nested within [ ].

/ (slash) indicates an either/or choice.



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

<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 memory. 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, non-null 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.

<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 alias name and field 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 an ASCII 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 with dBASE IV* to change Config.db parameters.

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

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

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.)

Using the substring comparison operator, you can test if one string is identical to or contained within a second string. You can search for the substring in a memory variable, in a character string, or in a memo field. For example, the following procedure looks for the character string "C-111-6045" in the Clie\_n\_hist memo field, and executes the Call\_rt routine if the string is found, or the Archive routine if it is not:

```
USE Client
DO WHILE .NOT. EOF()
  IF "C-11-6045" $ Clie_n_hist
    DO Call_rt
  ELSE
    DO Archive
  ENDIF
  SKIP
ENDDO
```

The AT() function and SUBSTR() function also work with substrings, but the \$ operator provides a basic test that is useful in many search routines.

## 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.

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.

---

## 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 run time, 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.

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 run time.

 **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 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. See also 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. A series of complex UDFs may run out of memory before you can reach the maximum nesting level.

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.

### Restrictions on Intervening Commands

UDFs 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 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, except \_pdriver which is often changed from the Config.db. You can change their values through the reports design and labels design screens, at the dot prompt, from the Config.db file, 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

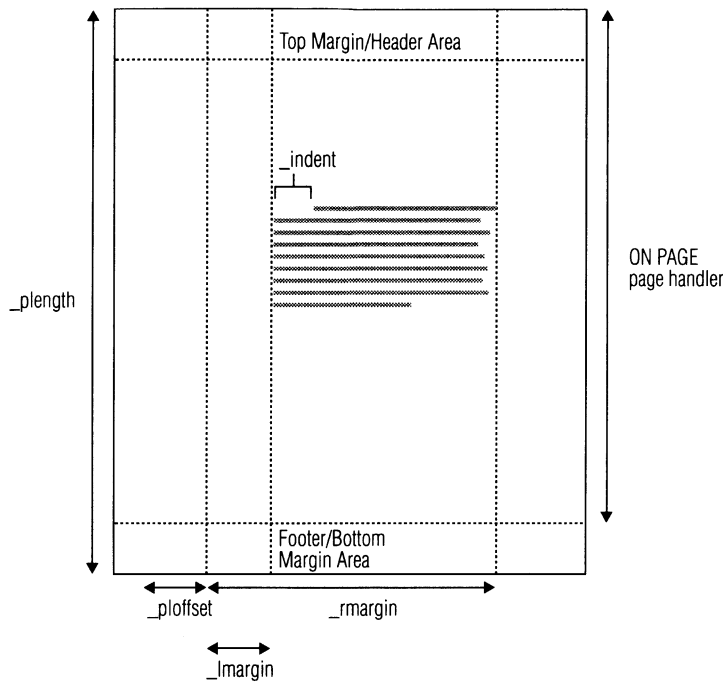


Figure 1-1 Page layout for typical printjob

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."

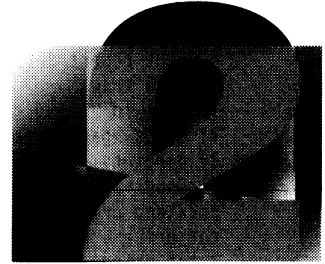
The reports design and labels design screens handle system variables for you. When you create a report (CREATE/MODIFY REPORT) or label (CREATE/MODIFY LABEL), you can store the changed system variable definitions to a binary *print form file*, which has a .prf extension. To change the default settings in this file, refer to *Using dBASE IV*. 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



# 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 ; mark 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 typesyles. In general, use `?/??` with the `STYLE` option to change typesyles 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.

 **TIP** 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”

---

## ???

??? 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 typesyles of individual text items, ??? to change typesyles on a broader basis within a document, and `_pscode` and `_pecode` to define the overall typestyle for a document.



**NOTE** ??? is the only way to output `CHR(0)` to the printer. `_pscode` and `_pecode` cannot do this.

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 below).

Table 2-1 Control character specifiers

---


<b>ASCII Code</b>	<b>Control Character Specifier</b>
0	{NULL} or {CTRL-@}
1	{CTRL-A}
2	{CTRL-B}
3	{CTRL-C}
4	{CTRL-D}
5	{CTRL-E}
6	{CTRL-F}
7	{BELL} or {CTRL-G}
8	{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}

---

*(continued)*

Table 2-1 Control character specifiers (continued)

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

 **TIP** 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. For all PostScript commands, be sure to include a space after each command; otherwise, you'll get no printed output. Use the following procedure:

1. Load the printer driver.
2. Send some output to the printer to initialize the .dld file.
3. After printing the database file, send an eject command with the "FF" string to eject the last page of output. For example:

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

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 download fonts to the printer, but proportional fonts will not line up correctly without a program. For example, to send 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(), SET PRINTER TO Chapter 5, "System Memory Variables"

## @

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

### Syntax

```
@ <row>, <col>
  [SAY <expression>
  [PICTURE <expC>] [FUNCTION <function list>]]
  [GET <variable>
  [[OPEN] WINDOW <window name>]
  [PICTURE <expC>]
  [FUNCTION <function list>]
  [RANGE [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, the third line from the bottom of 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 (UDF) 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 — With a color monitor, this option 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.

Except for SET COLOR to U and SET COLOR TO I, color settings will not appear on a monochrome system, although you may use the COLOR settings for applications that will be used with a color terminal.

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 the dBASE IV message **Editing condition not satisfied**. AT clause of the SET MESSAGE has no effect on the ERROR clause position.

**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. Use the REQUIRED keyword described below for specified range checking.

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 when you try to move the cursor into the **GET** field, whether you change the record or not. Without **REQUIRED**, the check occurs only when you change the **GET** field; changing other fields in the record does not recheck the **VALID** clause or **RANGE** parameters. The **REQUIRED** keyword must immediately follow **RANGE** or **VALID**, and only refers to the associated **RANGE** or **VALID** clause. For example, to make **REQUIRED** refer to both **RANGE** and **VALID** in a command line, use **RANGE REQUIRED** and **VALID REQUIRED**.

**WHEN** — You can provide a condition that is evaluated when you try to move the cursor out of the **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 format functions listed in Table 2-2.

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 in @...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 template symbols listed in Table 2-3.

Table 2-3 Template symbols

<b>Symbol</b>	<b>Accepts</b>
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


<b>Symbol</b>	<b>Description</b>
.	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
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.

## @ @...CLEAR

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**, **COL()**, **CREATE/MODIFY SCREEN**, **EDIT**, **MODIFY COMMAND**, **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**

---

## @...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.

@...CLEAR

@...FILL

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

### Example

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

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

### See Also

@...FILL, CLEAR

---

## @...FILL

@...FILL allows you to change the colors of a specific rectangular region on your color terminal 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.

@...FILL  
@...SCROLL

### 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

---

## @...SCROLL

@...SCROLL shifts the contents of a specified region of the screen up or down, or to the left or right.

### Syntax

```
@ <row1>,<col1> TO <row2>,<col2> SCROLL [UP/DOWN/LEFT/RIGHT]  
[BY <expN>] [WRAP]
```

### Defaults

@...SCROLL defaults to UP BY 1, without wrapping.

### Usage

<row1>,<col1> are the coordinates of the upper left corner of the region, and <row2>,<col2> are the coordinates of the lower right corner. The characters in this region of the screen are shifted up or down, left or right, by the number of rows or columns given in the BY option.

The WRAP option causes the characters that scroll off one edge of the region to replace the row or column on the opposite side that would otherwise be blank. For example, without WRAP, SCROLL LEFT BY 2 shifts two columns of characters off the left edge of the region and leaves two blank columns against the right edge. With SCROLL LEFT BY 2 WRAP, two columns of characters scroll off the left edge of the region and appear back on the right, as if characters wrapped around within the region.

If a window is active, row and column positions are relative to the active window.



The numeric expression in the BY option can be from -32,767 to +32,767. If you provide a negative number, the direction of the scrolling is reversed, so SCROLL LEFT BY -3 and SCROLL RIGHT BY 3 are the same. If <expN> is 0, no rows or columns are scrolled.

### See Also

@...TO, ACTIVATE WINDOW, DEACTIVATE WINDOW, DEFINE WINDOW

---

## @...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** Although @...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. Note that the ASCII code values (decimal 0 through 255) are listed in Appendix G. 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 quotation marks ( " "), 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.

## ACCEPT ACTIVATE MENU

### 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:

```
. 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]/[NOORGANIZE]

### 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  $\downarrow$  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 **Ctrl-End**. If you are using the dBASE IV 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.

NOORGANIZE brings up a menu bar without the **Organize** menu. **Organize** menu options to sort, index, and remove records are therefore unavailable.

 **TIP** 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>] [REINDEX] [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 use [[TYPE]<filename>] and 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 records in the Test database file that are not marked for deletion with the statement APPEND FROM Test FOR DELETED(), you will actually be testing the target file (the one currently open) for its deleted status. Therefore no records will be appended.

## 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 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 corner before using APPEND FROM.

REINDEX rebuilds any open non-controlling indexes after all new records to the database file are added. If you omit REINDEX, non-controlling indexes are updated after each record is added. The Options section in the BLANK command entry of this chapter discusses index updating, index rebuilding, and performance tuning with the REINDEX option.

In a multi-user environment, you can only use REINDEX if the database is opened for exclusive use.

 **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 can also use `IMPORT` to convert dBASE II, Framework II, Framework III®, Framework IV™, 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).

A file that was exported with `COPY TO...DELI WITH`, cannot be used as input for `APPEND FROM...DELI WITH` .

### 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

`BLANK`, `COPY`, `DELETED()`, `IMPORT`, `SET DELETED`

---

## APPEND FROM ARRAY

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

### Syntax

```
APPEND FROM ARRAY <array name> [REINDEX] [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. All field types, with the exception of the memo field, can be replaced with array information.

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.

If the array element and field have different data types, this command converts the element variable to the field's data type before writing it. If you do not want this data type conversion, verify that 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 new information to its elements, if you want values other than .F., before you can APPEND from it.

The REINDEX option rebuilds any open non-controlling indexes after all records have been changed. See the BLANK command entry in this chapter for more information.

In a multi-user environment, you can only use REINDEX if the database is opened for exclusive use.

### 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]    && There are 11 fields in the Travel database
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 reads 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.

If the existing memo field is larger than 64K, the message **Original memo cannot be larger than 64K** is displayed. If the existing memo is less than 64K, a file of any size can be appended.

When you use the OVERWRITE option, the existing memo field contents are deleted before new material is appended. With the OVERWRITE option, there's no restriction on the size of the existing memo field.

dBASE IV internal text editor MODIFY COMMAND cannot save memo fields that are greater than 64K, although it can open and let you view them.

## APPEND MEMO ASSIST

To edit memo fields greater than 64K, use an external program editor (such as Brief or Qedit) that does not have a 64K file size limit. Call the external editors with the WP setting in Config.db.

If you open a memo field with MODIFY COMMAND and cannot save your edited file, you can delete some text and try to save the file again.

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

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

### See Also

COPY MEMO, MODIFY COMMAND/FILE

---

## ASSIST

ASSIST gives you access to 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 *Getting Started with dBASE IV* and in *Using dBASE IV*. 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

The number of memory variables specified must be exactly the same as the number of fields AVERAGED.

The number of array elements must be at least the same as the number of fields AVERAGED.

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 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 program.

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.



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. If the transaction is incomplete use the ISMARKED() function to check the database.

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
- CREATE VIEW
- DELETE FILE
- ERASE
- INSERT
- MODIFY STRUCTURE
- PACK
- RENAME
- ZAP

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

## BEGIN/END TRANSACTION

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 INDEX TO
SORT...TO <newfile>
TOTAL...TO <newfile>
USE
```

If a command attempts to overwrite a file or close an open database or index file, which affects the transaction, an error message appears.

If you opened a file with the NOLOG option, it will not be part of the transaction log and it may be opened and closed without generating an error message or affecting the transaction log.

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 multi-user environment, 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
  .
  .
  .
  ENDSKAN
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(),  
USE

---

## BLANK

BLANK fills fields and records with blanks.

**Syntax**


```
BLANK [FIELDS <field list>/LIKE/EXCEPT <skeleton>]  
      [REINDEX] [<scope>] [FOR <condition>] [WHILE <condition>]
```

**Usage**

BLANK fills fields and records with blanks. The blank format is determined by the data type of the field specified, and the command used to view the data, as shown in Table 2-5.

Table 2-5 Displayed Values for Blank Data

Field Data Type	Browse/Edit	?	STORE	LIST	@SAY
Character					
Numeric		0	0		0
Float		0	0		0
Memo	memo			memo	memo
Date	//	//	//	//	//
Logical		.F.	.F.	.F.	

 **NOTE** Blank entries in Table 2-5 mean that nothing is displayed to the screen.

After a field or record is blanked, testing the field or record with the ISBLANK() function returns a logical true.

Using the BLANK command and the ISBLANK() function, you can control operations on blank data. For example, the commands

```
CALCULATE AVG(Salary)
```

and

```
BLANK ALL FOR Salary = 0  
CALCULATE AVG(Salary) FOR .NOT. ISBLANK(Salary)
```

do not return the same results if the Salary field in some records contains a zero. The first CALCULATE command sums all Salary fields in the database file and divides the sum by the number of records in the file. The second CALCULATE command divides the sum by the total number of records that actually contain salaries greater than zero. The first CALCULATE command assumes that a zero represents no salary, a literal zero. The second CALCULATE command assumes that zeroes are really nulls, unknown values.

BLANKing a record that previously contained data is similar to APPENDING a BLANK record. BLANK is also equivalent to the **Blank record** option in the **Records** menu of the EDIT and BROWSE commands. All three methods write a record that contains blank values. This cannot be done with commands such as REPLACE. The following commands do not fill a field that previously contained data with blanks:

```
REPLACE Salary WITH 0
REPLACE Switch WITH .F.
```

You can, however, blank out fields with the BLANK command, for example:

```
BLANK FIELDS Salary, Name, Switch
```

If you BLANK a field that is contained in the key expression of an index, dBASE IV must also adjust all open indexes that use this field as part of their key expressions. dBASE IV can either update the indexes, by changing only those entries in the index that are affected, or rebuild the index completely, as if you issued an INDEX ON command with the key expression.

Updates to the index are done after each record is BLANKed. Some forms of the BLANK command, such as using a FOR clause or using BLANK without a WHILE or <scope>, imply that only one record is changed. After the single change, all open indexes containing the blank field in the key expression are updated.

Rebuilding the index, on the other hand, is done only after multiple records are BLANKed. The rebuilding is done after all the changes are complete. Multiple records can be BLANKed by using a WHILE clause or providing a scope such as ALL.

Although many indexes may be open at one time, such as all the index tags contained in an open .mdx file, only one index controls the order of the records displayed. This index is the controlling index. Because the controlling index determines how records are displayed and accessed, dBASE IV treats this index differently than other, non-controlling indexes. Based on an optimization scheme, dBASE IV determines whether the controlling index should be updated or rebuilt. If it is updated, entries in the controlling index are rewritten after each record is changed. If it is rebuilt, all changes are made to the database file, and then the index file or tag is created anew.

Although you cannot state whether the controlling index should be updated or rebuilt, you can choose to rebuild the non-controlling indexes by using the REINDEX keyword of the BLANK command. Using REINDEX, all open, non-controlling indexes of the database open in the current work area that are affected by the BLANK command will be rebuilt once the BLANK command is complete.

You can optimize performance by using or omitting the REINDEX command. Typically, if you are changing only one or a few records, it is faster to omit REINDEX and allow the open, non-controlling indexes to be updated after each record is BLANKed. If you are changing many records, as with a BLANK NEXT 100, you might issue REINDEX, allowing the records to be BLANKed and the indexes to be rebuilt as a last step.

If you are developing an application, you may want to test commands that accept the REINDEX option for performance. Depending on the number of records in a file, the number of indexes, and the number of changes to the file, using or omitting the REINDEX keyword may speed up processing time.

### Special Case

You cannot BLANK any fields that are designated read-only, as with the PROTECT or USE NOUPDATE commands, or with the /R option of the BROWSE command.

In a multi-user environment, you can only use REINDEX if the database file is opened for exclusive use.

### See Also

ISBLANK()

---

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



**NOTE** *The /R and /<column width> options require that you enter the forward slash (/).*

## 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 **Unauthorized Access Level** message 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 *Getting Started with dBASE IV* and *Using dBASE IV*. 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. The BROWSE table is moved up to line 0, so you gain one extra display line.

NOORGANIZE brings up a menu bar without the **Organize** menu. **Organize** menu options to index, sort, and remove records are therefore unavailable. You cannot use both NOORGANIZE and NOMENU in the same BROWSE.

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.

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. It cannot be used with calculated fields.

<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.

# BROWSE

## Example

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:

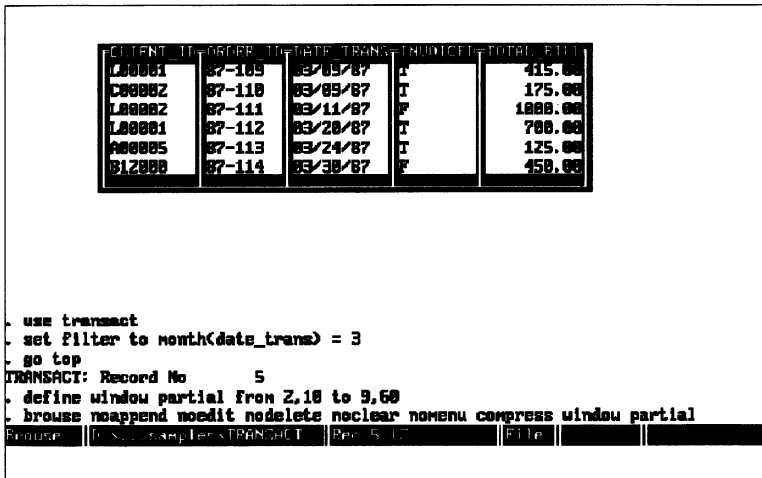


Figure 2-1 BROWSE window

## 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 contain any 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** Use the *ISBLANK()* function to test for blank values.

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() is always type F (floating).

STD(<expN>) calculates the standard deviation of the values in a database field. The formula for the standard deviation 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() is always type F (floating).

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 SUM(Total_bill), CNT() FOR Invoiced 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 first load the binary program files to memory with the LOAD command.

### Syntax

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

### Usage

The CALL command executes a binary program file that has been placed 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 is run from memory without having to be called from a disk. Up to 16 binary program files can be loaded in memory at one time, and each may be up to 65,500 bytes long.

When you CALL the binary program file, you specify the name of the file without the .bin extension. 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 parameters are passed as their character representation and end with a null (ASCII value of zero).

Refer to the LOAD command in this chapter for details on writing the binary program and for the passing of parameters to the .bin file.

### Options

The WITH option will accept 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 the contents of a field or a memory variable (or an array element), you can change its value.

### Example

See the examples for the LOAD command.

### See Also

CALL(), LOAD, RUN/!, RUN()

---

## CANCEL

CANCEL stops the execution of a program file, closes all open program 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.

In a transaction, CANCEL performs a ROLLBACK.

**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, RETURN, 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] [NOORGANIZE]
      [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 or currently active window, 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/SCREEN/
      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 and clears all pending GETs.

### Options

ALL closes all open database files and low level-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. 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 with dBASE IV* for memory limitations.)

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 SCREEN** command.

**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 SCREEN, 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/PRINTER/PROCEDURE
```

### Usage

**CLOSE ALL** closes all files of all types, including low-level files, and reselects work area 1. However, files opened by **SET DEVICE TO <filename>** or **SET PRINTER TO <filename>** will remain open.

## CLOSE COMPILE

To CLOSE a particular file type, follow the command with the keyword for the file type, as specified in the syntax.

CLOSE DATABASES 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.

CLOSE PRINTER closes the file opened by SET PRINTER TO FILE <filename>.

A file opened by SET DEVICE TO FILE <filename> will be closed when SET DEVICE is rerouted.

### 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 object code file (.dbo) that can be executed by dBASE IV.

### Syntax

```
COMPILE <filename> [RUNTIME]
```

## Defaults



**NOTE** All dBASE IV version 1.0 object code files must be recompiled in dBASE IV version 2.0 in order to run under version 2.0. 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 file was created with the DBLINK utility.

*Other versions of dBASE IV object code files need not be recompiled to run under version 2.0. Recompilation is recommended, however, if you want to take advantage of version 2.0 enhancements.*

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 create an 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.

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 the 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 have SET DEVELOPMENT OFF, 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. If SET TALK is OFF warning messages will be suppressed. If a macro appears in a command line the command line is expanded and parsed at execution time.

Make sure that each source code file has a unique name. If two source code 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>.

## Special Case

dBASE IV versions 1.5 and higher support conditional compiler directives in a procedure. The procedure may be in a .prg, .prs, .fmt, .lbg, .frg, or user-assigned file. Using these directives, you can indicate sections of the program that should not be compiled. These sections, and the compiler directives themselves, are not written in the .dbo file.

The compiler directives are:

```
#define <name>
#undef <name>
#ifdef <name>
#ifndef <name>
#else
#endif
```

You can define a name with `#define`, and then test if the name has been defined with `#ifdef`. You can also test if the name has not been defined with `#ifndef`. The `#else` and `#endif` directives complete the branching structure, like the `ELSE` and `ENDIF` keywords of the `IF` command. For example, you can include two versions of a menu procedure in a program, one designed for the UNIX version of dBASE IV, and the other designed for the DOS version of dBASE IV:

```
*Condcomp.prg
#define UNIXVER
#ifdef UNIXVER
    PROCEDURE Menu
        .
        .
        .
#else
    PROCEDURE Menu
        .
        .
        .
#endif
```

Names are local to the procedure that defines them, and, like memory variables, are also available to lower-level procedures. If you `#define` a name in a lower-level procedure, the name is not available to the calling procedure. Similarly, if you `#define` a name in one procedure and `#undef` it in a lower-level procedure, the name is unavailable to the lower-level procedure, but it is still available to the calling program that defined it.

Names can be up to 32 characters long and are not case sensitive. However, only the first nine letters are relevant.

Compiler directives can be nested to 32 levels.

### Optimization

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, PROCEDURE, SET LIBRARY, SET PROCEDURE, SET SYSPROC, 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 for storing multi-user lock information.

**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 `_dbaslock` 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 `_dbaslock` 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 `_dbaslock` field. It indicates who has locked the record or file, and when the lock was placed. If you place a file lock, the `_dbaslock` field of the first record in the database file contains the information used by `CHANGE()` and `LKSYS()`.

### Special Case

In a multi-user environment, the database file must be in exclusive use before you issue the `CONVERT` command. Make sure that `SET DELETED` is `OFF` before running `CONVERT`, or `REINDEX` the database after running `CONVERT`.

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

### See Also

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

---

## COPY

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

### Syntax

```
COPY TO <filename>  
  [[<TYPE>] <file type>]/[[<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, or the TYPE option DBMEMO3 is used. If you specify another TYPE option the contents of the memo field will not be copied and you may get an error message.

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. Type F fields are converted to type N fields.
- **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 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 Case

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.

 **TIP** 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 copies a file to a new filename.


### 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 path. You do not need to include the directory or drive designator of the first file if it is already established with the SET PATH or SET DIRECTORY command, the PATH setting, or in the Config.db file.

You cannot use COPY FILE to copy an open file.

 **TIP** *If you copy a database file that has memo fields, you must copy the associated memo (.dbt) file separately. You may use the COPY command to copy records from an open database file. Use the COPY TO... WITH PRODUCTION option if you wish to copy and rename .mdx files.*

### Example

To make a duplicate of a dBASE program file:

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

### See Also

COPY, SET DIRECTORY, 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 Case

In a multi-user 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 the path.

### 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 `Clie_n_hist` to a file named `Cus_text`, appending the information in the current record to information that already exists in the file:

```
. COPY MEMO Clie_n_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 .dnt 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 \_dbaslock 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.

## 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. The FOR clause of an .mdx file tag will be ignored, as it is not supported by .ndx files.

## Example

To copy the Order\_id tag from the Stock .mdx file to an .ndx file called Items\_id:

```
. USE Stock
. COPY TAG Order_id TO Items_id.ndx
  100% indexed          17 records indexed
. SET INDEX TO Items_id
  Master index: ITEMS_ID
. DISPLAY STATUS
Currently Selected Database:
Select area: 1, Database in Use: C:\SET\STOCK.DBF Alias: CLIENT
                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, DESCENDIN(), FOR(), INDEX, MDX(), NDX(), SET INDEX, SET ORDER, TAG(), TAGCOUNT(), TAGNO(), UNIQUE

---

## 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 are no more rows in the array.

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

### Options

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

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

### Example

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

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

If **RECCOUNT()** exceeds 65,535 records, the error message **Bad array dimensions** appears. This is because each array dimension is limited to 65,535 elements.

### See Also

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

## COUNT

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

### Syntax

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

### Usage

If SET TALK is ON, this command counts the number of records and displays the tally. If you specify a condition with a FOR or WHILE clause, or limit the number of records with a scope, the tally indicates the number of records that meet the condition or fall within the scope. COUNT can be limited to records according to 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 multi-user environment COUNT automatically locks the file during its operation if SET LOCK is ON (the default), and unlocks it after the count is complete. If SET LOCK is OFF, a COUNT can still be performed, but the result may not be reliable if another user is changing the database file.

### Example

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

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

### See Also

AVERAGE, CALCULATE, RECCOUNT(), SUM

---

## CREATE or MODIFY STRUCTURE

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

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>

or

MODIFY STRUCTURE

### Defaults

Unless you specify a directory with the filename, CREATE writes the new database file in the default 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 (Character, Numeric [Binary Coded Decimal], Floating point numeric, Logical, Date, or Memo), 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.

 **TIP** *MODIFY STRUCTURE makes backup copies of the database file (.dbk extension), memo file (.tbk extension), and multiple index file (.mbx extension) in the same directory as the original files. 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.*

*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 multi-user 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 used by SQL 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 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** *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.*

## CREATE FROM CREATE/MODIFY APPLICATION

### 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 2-6 with the Applications Generator.

Table 2-6 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 directory.

## 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 dBASE IV* 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 or from the design surface directly 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 dBASE IV* 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 by earlier versions of dBASE programs. 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 dBASE IV* 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 .fmg 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 dBASE IV* 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 if you set FORMAT TO the resulting .fmt file.

### 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 quotation 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 dBASE IV* contains further information on using the screen designer.



## CREATE VIEW FROM ENVIRONMENT

CREATE VIEW FROM ENVIRONMENT builds a view (.vue) file that is compatible with dBASE III PLUS, if your dBASE IV environment does not exceed dBASE III PLUS limits.

### 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 or work areas:

- 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
- SET KEY setting

## CREATE VIEW FROM ENVIRONMENT DEACTIVATE MENU

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.

## DEACTIVATE POPUP DEACTIVATE WINDOW

```
. 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. DEACTIVATING 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, shows 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 can 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 can enter dBASE expressions on the left of this window, but not expressions using the macro substitution character (&). 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.
- **U** Suspend — Temporarily exit the debugger to the dot prompt. From the dot prompt, type **RESUME** to return to the debugger. Type **CANCEL** to end the debugger session and remain at the dot prompt.

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 ten constants. The number of variables is 50.

### 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 limited only by your system's memory. A single dimension, however, is limited to 65,535 elements.

```
. DECLARE Cost[15]
```

creates a one-dimensional array (a row).

## DECLARE

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. You can pass individual elements of an array to procedures and functions, but you cannot pass the entire array by reference. 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 with dBASE IV* 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 popup. 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. You can define up to 16,378 bars.

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 SKIP the BAR 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 is 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.

## DEFINE BOX

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 IV.

### 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
CLEAR
USE Stock ORDER Part_name
_box = .F.
DEFINE BOX FROM 10 TO 70 HEIGHT 8
?? SPACE(51) AT 15
_box = .T.
?
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:

```

.do box
BOOKCASE          WOOD, TEAK, 2-SHELF
CHAIR, DESK      LEATHER, BROWN, HIGHBACK
CHAIR, DESK      LEATHER, BROWN, HIGHBACK
CHAIR, DESK      LEATHER, BROWN
CHAIR, DESK      LEATHER, BROWN
CHAIR, SIDE      PLASTIC, GREY
DESK, EXECUTIVE 5-FOOT WOOD, OAK, FANCY
FILE CABINET, 2 DRAWER METAL, BROWN
FILE CABINET, 2 DRAWER METAL, BLACK
FILE CABINET, 4 DRAWER METAL, BROWN
LAMP, FLOOR      BRASS, 6-FOOT, ENGLISH
LAMP, FLOOR      BRASS, 6-FOOT, ART DECO
LAMP, FLOOR      BRASS, 6-FOOT, ENGLISH
SOFA, 6-FOOT     LEATHER, BROWN, HIGHBACK
SOFA, 6-FOOT     VELVET, GREY, FRENCH
SOFA, 8-FOOT     VELVET, BLUE, FRENCH
TABLE, END       WOOD, OAK, 2-FOOT, SQUARE

```

Command | D:\ndbase\samples\STOCK | Rec E0F17 | File |

Figure 2-2 Boxed output

**See Also**

@...TO, SET BORDER, \_box

Chapter 5 describes the 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.

## DEFINE MENU

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` 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

\* 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
```

### 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.

## DEFINE PAD

### 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 popups. The following program example creates a menu named Main and its four pads, with popups for each pad and a bar for the first popup.

```
* 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 and the line above the status bar. If the status bar is off, the last line is the screen size minus 1. The minimum size for a popup 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 ↵.

## DEFINE POPUP DEFINE WINDOW

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 (up to 200) 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 an inverse video panel, define a double-line box, 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 **Print Screen** 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. If you do not specify COLORS, dBASE IV defaults to the Config.db specifications.

You can store up to 20 window definitions in memory at one time depending on the amount of memory available.

**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(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 Case

In a multi-user 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.

## DELETE TAG DEXPORT

### Example

To delete the Client tag from the Client production .mdx file:

```
. DELETE TAG Client OF Client
```

### See Also

COPY INDEX, COPY TAG, DESCENDING(), FOR(), INDEX, MDX(), NDX(), ORDER(), SET INDEX, SET ORDER, TAG(), TAGCOUNT(), TAGNO(), UNIQUE, USE, SET EXCLUSIVE

---

## DEXPORT

This command creates a Binary Named List (BNL) file from a screen, report, or Label design file.

### Syntax

```
DEXPORT SCREEN/REPORT/LABEL <filename> [TO <BNL filename>]
```

### Options

<filename> is the name of the dBASE design object file.

<BNL filename> is an optional name for the output file. If omitted, DEXPORT uses the root name of the design file and adds the file extensions listed in Table 2-7 to BNL files.

Table 2-7 Binary Named List file extensions

---

Design File	BNL File Extension
Screen	.snl
Report	.fnl
Label	.lnl

---

### See Also

DGEN()

See *Programming in dBASE IV* for more information on dBASE IV generated code.

## 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. These may have been modified by SET DEFAULT or SET 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
```

will show a file called Samples.dbf if it exists. 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\*.dbf
```

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 piece of information is presented at a time, and a prompt may appear 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.

LIST without an argument uses ALL as its default scope. DISPLAY without an argument uses the current record only.

Please refer to the following LIST commands for a discussion of both the LIST and DISPLAY forms:

- LIST/DISPLAY FILES
- LIST/DISPLAY HISTORY
- LIST/DISPLAY MEMORY
- LIST/DISPLAY STATUS
- LIST/DISPLAY STRUCTURE
- LIST/DISPLAY USERS



## DO

DO executes a dBASE command file or procedure. If the command file or procedure file has not been COMPILED, it is first parsed, then compiled and saved as an object file with a .dbo extension; then, the .dbo file is executed.

DO may also pass parameters to the named program.

### Syntax

```
DO <program filename>/<procedure name>
  [WITH <parameter list>]
```

### Usage



**NOTE** All dBASE IV version 1.0 object code files must be recompiled in dBASE IV version 2.0 in order to run under version 2.0. 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 file was created with the DBLINK utility.

Other versions of dBASE IV object code files need not be recompiled to run under version 2.0. Recompilation is recommended, however, if you want to take advantage of version 2.0 enhancements.

The filename must include a path, if the file is not on the current directory or on a path set with the SET PATH or SET DIRECTORY command.

You should not, however, precede the names of procedures within a procedure file with a path. The 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. Look in the file SYSPROC = <filename> if specified in Config.db.
2. DO searches for a procedure in the current .dbo file, if one is being executed.
3. 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.
4. DO searches for a procedure in other open .dbo files.
5. Look in the SET LIBRARY file if active.
6. DO searches for a .dbo file with the associated name.
7. DO searches for a .prg file with the associated name, compiles it, and then executes the resultant .dbo file.

8. Look for a .prs file of that name.

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. 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 can 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 and FILES setting in Config.db. The lower of the two settings will prevail. DOS reserves five file handles for its use. To have 100 files open you need to set the FILES= setting to 99 in Config.sys. The larger the number, the less room there is in memory for programs and data.

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, 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->*.

 **TIP** 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  $\text{area} = \text{length} * \text{width}$ :

```
* 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 LIBRARY, SET PROCEDURE, SET TRAP, SUSPEND

Chapter 15 of *Programming in dBASE IV*

## DO CASE/ENDCASE

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  $\text{Numvar} < 11$ , 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. DO CASE can compile correctly with no CASE or no OTHERWISE statements.

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.).

 **TIP** *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 are a number of exceptions to a condition. The CASE <condition> statements can represent the exceptions, and the OTHERWISE statement the remaining situation.

## DO CASE/ENDCASE DO WHILE/ENDDO

### 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/ENDDO

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.

## DO WHILE/ENDDO EDIT

```
* Program name: Partial.prg
USE Stock
DO WHILE .NOT. EOF()
CLEAR
  LIST NEXT 5 Order_id, Part_name, Item_cost
  ?
  WAIT "Press X to stop, B to back up, Spacebar to continue." TO Mstop
DO CASE
  CASE Mstop $ "Xx" .OR. EOF()
    EXIT
  CASE Mstop $ "Bb"
    SKIP -9
  OTHERWISE
    SKIP
ENDCASE
ENDDO
* EOP: Partial.prg
```

### See Also

&, DO, DO CASE, IF, RETURN, SCAN

---

## EDIT

EDIT is a full-screen command you use to display or change the contents of a record in the active database file or view.

### Syntax

```
EDIT [NOINIT] [NOFOLLOW] [NOAPPEND] [NOMENU] [NOORGANIZE]
     [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 limited only by any conditions or SET FILTER.

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.

NOORGANIZE brings up a menu bar without the **Organize** menu. The **Organize** menu options to index, sort, and remove records are therefore unavailable. You cannot use both NOORGANIZE and NOMENU in the same EDIT.

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 Case**

In a multi-user environment, if the file is not opened in EXCLUSIVE mode and 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 2-8 Navigation keys

<b>Key</b>	<b>Action</b>
<b>F1 Help</b>	Help
<b>F2</b>	Toggle BROWSE/EDIT
<b>F3</b>	Previous field
<b>F4</b>	Next field
<b>F5-F9</b>	No action
<b>F10</b>	Menu
<b>Shift-F2</b>	Transfer to query design
<b>Shift-F3</b>	Repeat find, backward
<b>Shift-F4</b>	Repeat find, forward
<b>Shift-F5 – Shift-F7</b>	No action
<b>Shift-F8</b>	Copy same field from previous record
<b>Shift-F9</b>	Print menu
<b>Shift-F10</b>	Macro (keyboard) processing

**See Also**

BROWSE, CHANGE, CONVERT, 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.

 **TIP** *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.

## EJECT PAGE ERASE

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.

 **TIP** You may EJECT PAGE before the output reaches the ON PAGE line in order to call the page handler. The page handler should take care of the page eject, and may optionally write a footer on the current page and a header at the top of the next page.

Use EJECT, not EJECT PAGE, to simply eject a page on the printer. The EJECT command does not affect the `_pageno` and `_plineno` system variables, although it does honor `_padvance` and `_pwait`.

### See Also

???, EJECT, ON PAGE, SET ALTERNATE, SET PRINTER, `_padvance`, `_pageno`, `_plength`, `_plineno`  
Chapter 1, "Essentials," discusses system memory variables and streaming output.

---

## ERASE

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.

**Usage**

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

Use ERASE ? to display a list of files in all directories.

You may not delete an open file.

To erase a file in another directory, you must explicitly state the path in the filename or use ERASE ? to select files from a file list.

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.

**See Also**

CLOSE, 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, Framework III, Framework IV, RapidFile, or Lotus 1-2-3.

**Syntax**

```
EXPORT TO <filename> [TYPE] PFS/DBASEII/FW2/FW3/FW4/  
RPD/WKS/WK1 [FIELD <field list>] [<scope>]  
[FOR <condition>] [WHILE <condition>]
```

**Usage**

EXPORT creates files that can only be used by PFS:FILE, dBASE II, Framework II, Framework III, Framework IV, RapidFile, or Lotus 1-2-3. 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.

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
```

## FIND

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:

```
. FIND "&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 ten 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. 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.

**M 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.

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 attributes 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 attributes of the window inside the interrupting UDF, when you return, the window will be in the new location and display the new attributes.

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.*

### Restoring the Record Pointer

The record pointer tells dBASE IV which record is active. If executing an interrupt routine that moves the record pointer, dBASE doesn't restore the original record pointer even if you set DBTRAP on. In every work area, whichever record is active at the end of an interrupt remains the active record.

To ensure that an interrupted process (such as BROWSE or APPEND) resumes smoothly after an interruption, your interrupt functions or procedures should include steps to restore all record pointers to their original values. You can use the RECNO() function to determine the value of the record pointer, save that value, then reset it with the GOTO command after the interruption, as shown in the following example:

```
FUNCTION myudf
oldrecno = RECNO()
[the rest of your interrupt routine]
GOTO oldrecno
RETURN
```

### Replacing APPEND Values

If you interrupt an APPEND with a routine that moves the record pointer, you should take care to restore the record pointer to its original value. For more information, read the guidelines described in the previous section, "Restoring Record Pointers."

In addition, you should save all values entered prior to your interrupt routine and restore those values after repositioning the record pointer. These steps are required because dBASE IV keeps all values entered during the APPEND command in a temporary buffer. During APPEND, the buffer record values are handled as follows:

- If you press **Esc** and abandon the new record, dBASE discards the values.
- If you accept the new record during a normal APPEND, the values are added to the database file as a permanent record.
- If you move the record pointer during an APPEND interrupt, dBASE discards the values in the temporary buffer.

To ensure that an APPEND operation resumes smoothly after an interruption, use the following guidelines when writing your function:

1. Save the current record pointer
2. Save any values already entered.
3. Execute the interrupt routine.
4. Restore the original record pointer.
5. Restore the previously-entered record values.

### 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.

### & 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
```

To use the `Plus_tax` UDF from the dot prompt, type the following:

```
. SET TALK OFF
. SET PROCEDURE TO Plus_tax
. M_tax = .10
. ? Plus_tax(100, M_tax)
    110
```

### See Also

DO, PARAMETERS, PROCEDURE, RETURN, SET DBTRAP, SET LIBRARY, SET PROCEDURE

Chapter 1 of this manual and Chapter 15 of *Programming in dBASE IV*, which contains information on the `SYSPROC` setting.

## 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 40
- 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 40 (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 multi-user 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/ENDIF

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
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, Framework III, and Framework IV database and spreadsheet frames, RapidFile data files, and Lotus 1-2-3 spreadsheets.

**Syntax**

```
IMPORT FROM <filename> [TYPE] PFS/DBASEII/FW2/FW3/FW4/
RPD/WK1/WKS
```

**Defaults**

The filename must include the file extension, if one exists and is different than the default extension: dBASE II database files have a .dbf extension; Framework II, Framework III, and Framework IV files an .fw2, .fw3, or .fw4 extension, respectively; RapidFile data files an .rpd extension; and Lotus 1-2-3 files have a .wks or .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 in 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, a compiled format (.fmo) file, and a view (.vue) file. All four 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> [UNIQUE]
or
INDEX ON <key expression>
    TAG <tag name> [OF <.mdx filename>]
    [FOR <condition>] [UNIQUE] [DESCENDING]
```

**Defaults**

Unless you specify otherwise as part of the filename, the default drive and current 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 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 dBASE filenames; they can be up to eight characters long, and can 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.

 **TIP** *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.

If you create a conditional index with INDEX ON <key expression> FOR DELETED(), RECALL ALL will only recall one record even if there are many deleted records. The best way to perform RECALL ALL is to close the master controlling index.

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 file is created, the index (.ndx) file is added to the catalog. The .mdx file and any index tags contained within an .mdx file are not, however, added to the catalog.

In a multi-user 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 \$1,000 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**

ALIAS(), CLOSE, COPY INDEX, COPY TAG, DELETE TAG, DESCENDING(), FIND, FOR(), KEY(), LIST STATUS, LOOKUP(), MDX(), NDX(), TAGCOUNT(), ORDER(), REINDEX, SEEK, SEEK(), SET DELETED, SET EXCLUSIVE, SET FILTER, SET INDEX, SET NEAR, SET ORDER, SET UNIQUE, SORT, STR(), SUBSTR(), TAG(), TAGNO(), UNIQUE()

---

## 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]/[NOORGANIZE]

### 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. This command is not recommended for extremely large files.

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 Help** 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 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. NOORGANIZE brings up a menu bar without the **Organize** menu. The **Organize** menu options to sort, index, and remove records are therefore unavailable.



**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

INSERT requires exclusive use of the database file in a multi-user environment.

### 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.

 **TIP** 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 adequate disk space allocated when JOINing two files. Two database files can be JOINed such that the new file exceeds the available 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.

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 KEYBOARD

```
. 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: cdbase\samples-ewfile.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		N
2	CLIENT	Character	30		N
3	DATE_TRANS	Date	8		N
4	TOTAL_BILL	Numeric	8	2	N
5	ORDER_ID	Character	6		N
**	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, mnemonic strings, or key labels into the type-ahead buffer. dBASE IV then reads the characters as if the user typed them in.

### Syntax

```
KEYBOARD <expC> [CLEAR]
```

### Options

<expC> can be any character or character string, the CHR() function, or any key label listed in Supported key labels table on next page.

CLEAR clears the keyboard buffer before inserting new characters. If the clear option is omitted, the buffer is not cleared.

**Usage**

dBASE reads the characters in the keyboard buffer when you execute a command that expects keyboard input, such as READ, BROWSE, or EDIT. KEYBOARD truncates any characters that exceed the keyboard buffer as determined by the SET TYPEAHEAD command. The keyboard buffer can be set to any value from 0 to 250. The default is 20.

Specify <expC> in any of the following ways:

Character string	Use dBASE delimiters: ", ', or []
CHR() function	Use no delimiters. 1 through 255 are supported. CHR(0) is ignored.
Key Labels	Use both delimiters and curly braces around the exact key label. For example, "{DNARROW}"
Numeric ASCII	Use both delimiters and curly braces around the numeric ASCII value. Also use the Alt key and the numeric keypad to enter special ASCII characters. For example, [{89}]

Supported key label names must be entered as shown in the following table. Key labels are not case-sensitive.



**NOTE** You cannot use *Enter* or *Esc* with the *KEYBOARD* command. To send an *Enter* key, use either *Ctrl-M* or {13}. To send an *Esc* key, use {27}. You can continue to use the *CHR()* function with these values.

Table 2-9 Supported Key Labels

<b>Alt-0 through Alt-9</b>	<b>Ctrl-PgDn</b>	Leftarrow
<b>Alt-A through Alt-Z</b>	<b>Ctrl-PgUp</b>	<b>PgDn</b>
<b>Backspace</b>	<b>Ctrl-Rightarrow</b>	<b>PgUp</b>
<b>Backtab</b>	<b>Del</b>	Rightarrow
<b>Ctrl-A through Ctrl-Z</b>	<b>Dnarrow</b>	<b>Shift-F1 through Shift-F9</b>
<b>Ctrl-End</b>	<b>End</b>	<b>Tab</b>
<b>Ctrl-F1 through Ctrl-F10</b>	<b>F1 through F10</b>	Uparrow
<b>Ctrl-Home</b>	<b>Home</b>	
<b>Ctrl-Leftarrow</b>	<b>Ins</b>	

**See Also**

CHR(), CLEAR TYPEAHEAD, INKEY(), READKEY(), 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.

### Syntax

```
LABEL FORM <label filename>/?  
    [<scope>] [FOR <condition>] [WHILE <condition>]  
    [SAMPLE] [TO PRINTER/TO FILE <filename>]
```

### Defaults

Unless the file is in the current directory or a path is set, the filename must include the 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 file, rather than to the printer or screen. If you direct the output to a 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 Case

In a multi-user environment 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 and \_pform, which affect printed output.

*Using dBASE IV* discusses the label design screen 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 shows all records, unless limited by the scope, a FOR or WHILE clause, SET FILTER, SET KEY, 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**. 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.

### 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.

### Example

```
. LIST FILES

Database Files    # Records    Last Update    Size
CLIENT.DBF       8            11/05/87       1570
STOCK.DBF        17           11/05/87       1569
TRANSACTION.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 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.

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.

## LIST/DISPLAY MEMORY

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 Mgrereetings; the Binary Coded Decimal 400.34 has been stored to MFLOAT; a logical true (.T.) has been stored to Mtruth; and the binary 3.21E+06 has been stored to the memory variable MFIXED.

```
. 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 with dBASE IV* contains information on Config.db settings, including MVBLKSIZE, MVMAXBLKS, RTBLKSIZE, and RTMAXBLKS.

## LIST/DISPLAY STATUS

LIST/DISPLAY STATUS provides information about the current dBASE IV session.

### Syntax

LIST/DISPLAY STATUS [TO PRINTER/TO FILE <filename>]

### Usage

DISPLAY STATUS pauses after each screen's display and prompts you to **Press any key to continue...** LIST STATUS scrolls without pausing.

For each open database file, LIST/DISPLAY STATUS shows the following information:

- Current work area number
- Database name with disk, 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

If any low-level files are open it shows:

- The file handle number
- The filename
- The file's open mode
- The current position in bytes from the beginning of the file
- The size of the file In addition, it shows:
- File search path
- OS working drive/directory
- Print destination
- Loaded modules
- Currently selected work area
- Left margin setting
- Currently open PROCEDURE file
- Reprocess count

## LIST/DISPLAY STATUS LIST/DISPLAY STRUCTURE

- 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
- Currently open library file
- Currently open system procedure (SYSPROC) file
- Current key range

### Options

TO PRINTER directs command output to the printer, and TO FILE directs the output to a file on disk.

### Special Case

In a multi-user 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 screen (whether the active screen is a small window or full display screen), then DISPLAY STRUCTURE pauses after each screen 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 and extracts the network-assigned names of the workstations currently logged in.

The command looks for Login.db according to the path specified by CONTROLPATH, and then in the directory where Dbase.exe resides.

### **Special Cases**

Always use LIST/DISPLAY USERS, or its network operating system equivalent, to determine whether anyone is using dBASE IV before it is uninstalled.

If two or more users log in with the same workstation name, this command displays the user name only once. For example, if two users log in as WKSTN1, LIST/DISPLAY USERS shows:

**Computer Name**

>WKSTN1

even if they both logged in from the same directory.

### **See Also**

LIST/DISPLAY STATUS, NETWORK()

---

## **LOAD**

LOAD allows you to load binary program files in memory.

### **Syntax**

LOAD <binary filename>

### **Usage**

LOAD places a binary (.bin) file in memory where it can be executed with the CALL command or the CALL() function.



## 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 65,500 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 original 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 Borland Turbo Assembler (TASM).

```
TASM <source>;  
TLINK <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 with the sample files, 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 3rd 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. The SEEK command can only be used if you are searching for the index key. SEEK searches through the .ndx or the tag. Such index files are usually much smaller than the database file.

Locate tests each record. 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

LOGOUT logs out the current user and sets up a new log-in screen when used with PROTECT. 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 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 Case

If PROTECT has not been used, no Dbssystem.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 directory or file extension as part of the filename for MODIFY COMMAND, the default 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 the dBASE IV 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**.

MODIFY COMMAND/FILE 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, although this is usually limited by available disk space.

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 with dBASE IV* 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. Each time that you edit a file, a backup copy of the previous version is saved in the same directory as the original file. The name of a backup file is the same as that of 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.

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 dBASE IV*.

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

---

## MOVE WINDOW

The MOVE WINDOW command moves a window to a new location on the screen.

### Syntax

```
MOVE WINDOW <window name> TO <row>,<column>/BY  
    <delta row>,<delta column>
```

### Usage

The syntax shown combines two different ways of moving a window. You can give the new coordinates for the window, or you can give the change you want in the placement of the window, relative to its current position.

Once you move a window, the coordinates of the new location are associated with that window name. If you want the window back at its original location, you must issue another MOVE WINDOW command that contains the original row and column parameters.

If the window does not fit on the screen at the new location, an error message appears and the MOVE WINDOW command does not take effect.

### Examples

To move a window called W1 to the right by 2 columns and down by 5 lines:

```
. MOVE WINDOW W1 BY 5,2
```

To move the window W1 to new coordinates:

```
. MOVE WINDOW W1 TO 10,5
```

### See Also

ACTIVATE WINDOW, DEACTIVATE WINDOW, DEFINE WINDOW, RESTORE WINDOW, SAVE WINDOW, SET WINDOW OF MEMO



---

## NOTE

NOTE, an asterisk (\*), or double ampersand (&&) characters indicate comment lines in a program (.prg) file.

### Syntax

NOTE/\* <text>

and

[<command>] &&<text>

### Usage

NOTE/\*&& inserts comments into program files for documentation and explanatory purposes. You can use any of the three forms 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 BAR

ON BAR executes a specified command when the user selects (highlights) a bar in a pop-up menu.

### Syntax

ON BAR <expN> OF <popup name> [<command>]

 **NOTE** *<command>* is any command except one that changes the flow of control, such as *IF*, *ELSE*, *DO WHILE*, and so on.

### Usage

Use ON BAR to execute a specified command when the user *selects* (highlights) a particular bar in a pop-up menu. A user can select a bar in the following ways:

- Use the ↑ and ↓ keys to move the cursor to the bar.
- Press the first letter of the bar's prompt.
- Click the bar.
- Drag the mouse cursor to the bar.

To execute a command when the user *chooses* (by pressing **Enter** or double-clicking the mouse) a bar, use ON SELECTION BAR. To execute a command when the user moves the cursor off a bar, use ON EXIT BAR.

To trigger the same command on *all* bars, use ON POPUP instead of ON BAR.

If you omit *<command>* when you use ON BAR, the command previously assigned for the specified menu bar is disabled.

### Example

In the following example, dBASE lists all of the database files in the current directory when the user highlights bar 1, and lists all of the .QBE files in the current directory when the user highlights bar 2.

```
DEFINE POPUP P1 FROM 1,1
DEFINE BAR 1 OF P1 PROMPT "Database Files"
DEFINE BAR 2 OF P1 PROMPT "Query Files"
ON BAR 1 OF P1 DO Showfiles WITH PROMPT()
ON BAR 2 OF P1 DO Showfiles WITH PROMPT()
ACTIVATE POPUP P1
RETURN
```

```
* Procedure =====  
PROCEDURE Showfiles  
PARAMETERS prompt  
DEFINE WINDOW Showfiles FROM 5,0 TO 20,79  
ACTIVATE WINDOW Showfiles  
DO CASE  
  CASE prompt = "Database Files"  
    DIR  
    WAIT  
  
  CASE prompt = "Query Files"  
    DIR *.QBE  
    WAIT  
ENDCASE  
DEACTIVATE WINDOW Showfiles  
RETURN
```

### See Also

BAR(), BARCOUNT(), BARPROMPT(), DEFINE BAR, DEFINE MENU, DEFINE PAD, DEFINE POPUP, MENU(), ON EXIT BAR, ON EXIT MENU, ON EXIT PAD, ON EXIT POPUP, ON MENU, ON PAD, ON POPUP, ON SELECTION BAR, ON SELECTION MENU, ON SELECTION PAD, ON SELECTION POPUP, PAD(), POPUP(), PROMPT()

---

## ON ERROR/ESCAPE/KEY

ON ERROR/ESCAPE/KEY suspends the current program and performs the specified action 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. 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.

If a dBASE IV error is not one of these two types, ON ERROR will not trap it. dBASE error 39, "Numeric Overflow", for example, is warning you of a mathematical result, and that you should adjust your code. 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 2-10 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 2-10 <key label name> descriptions for non-printing keys

Keycap Identification	<key label name>
<b>F1 to F10</b>	F1, F2, F3
<b>Ctrl-F1 to Ctrl-F10</b>	Ctrl-F1, Ctrl -F2
<b>Shift-F1 to Shift-F9</b>	Shift-F1, Shift-F2
<b>Alt-0 to Alt-9</b>	Alt-0, Alt-1, Alt-2

(continued)

Table 2-10 <key label name> descriptions for non-printing keys (continued)

<b>Alt-A to Alt-Z</b>	Alt-A, Alt-B, Alt-C
←	LEFTARROW descriptions for non-printing keys
→	RIGHTARROW
↑	UPARROW
↓	DNARROW
<b>Home</b>	Home
<b>End</b>	End
<b>PgUp</b>	PgUp
<b>PgDn</b>	PgDn
<b>Del</b>	Del
<b>Backspace</b>	BACKSPACE
<b>Ctrl-←</b>	Ctrl-leftarrow
<b>Ctrl-→</b>	Ctrl-rightarrow
<b>Ctrl-Home</b>	Ctrl-Home
<b>Ctrl-End</b>	Ctrl-End
<b>Ctrl-PgUp</b>	Ctrl-PgUp
<b>Ctrl-PgDn</b>	Ctrl-PgDn
<b>Ins</b>	INS
<b>Tab</b>	TAB
<b>Back Tab</b>	BACKTAB
<b>Ctrl-A to Ctrl-Z*</b>	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 LIBRARY, SET PROCEDURE, SET TYPEAHEAD, UPPER(), WAIT

---

## ON EXIT BAR

ON EXIT BAR executes a specified command when the user moves the cursor off the bar in a pop-up menu.

### Syntax

ON EXIT BAR <expN> OF <popup name> [<command>]

 **NOTE** <command> is any command except one that changes the flow of control, such as IF, ELSE, DO WHILE, and so on.

### Usage

Use ON EXIT BAR to execute a specified command when the user moves the cursor off a particular bar in a pop-up menu.

A user can move the cursor off a bar in the following ways:

- Use the ↑ and ↓ keys to move the cursor to a different bar.
- Press the first letter of another bar's prompt.
- Click or double-click another bar.
- Drag the mouse cursor from the bar.
- Deactivate the pop-up menu.

To execute a command when the user *selects* (highlights) a bar, use ON BAR. To execute a command when the user *chooses* a bar, use ON SELECTION BAR.

To trigger the same command on *all* bars, use ON EXIT POPUP instead of ON EXIT BAR.

If you omit <command> when you use ON EXIT BAR, any command previously assigned for the specified bar is disabled.

### Example

In the following example, dBASE IV lists all of the database files in the current directory when the user moves the cursor away from bar 2, and lists all of the .QBE files in the current directory when the user moves the cursor away from bar 3.

```
DEFINE POPUP P1 FROM 1,1
DEFINE BAR 1 OF P1 PROMPT "Files"
DEFINE BAR 2 OF P1 PROMPT "Database Files"
DEFINE BAR 3 OF P1 PROMPT "Query Files"
ON EXIT BAR 2 OF P1 DO Showfiles WITH PROMPT()
ON EXIT BAR 3 OF P1 DO Showfiles WITH PROMPT()

ACTIVATE POPUP P1
RETURN
```



```
* Procedures =====
PROCEDURE Showfiles
PARAMETERS prompt
DEFINE WINDOW Showfiles FROM 5,0 TO 20,79
ACTIVATE WINDOW Showfiles
DO CASE
  CASE prompt = "Database Files"
    DIR
    WAIT

  CASE prompt = "Query Files"
    DIR *.QBE
    WAIT
ENDCASE
DEACTIVATE WINDOW Showfiles
RETURN
```

### See Also

BAR(), DEFINE BAR, DEFINE MENU, DEFINE PAD, DEFINE POPUP, MENU(), ON BAR, ON EXIT MENU, ON EXIT PAD, ON EXIT POPUP, ON MENU, ON PAD, ON POPUP, ON SELECTION BAR, ON SELECTION MENU, ON SELECTION PAD, ON SELECTION POPUP, PAD(), POPUP(), PROMPT()

---

## ON EXIT MENU

ON EXIT MENU executes a specified command when the user moves the cursor off certain pads in a specified menu.

### Syntax

ON EXIT MENU <menu name> [<command>]



**NOTE** <command> is any command except one that changes the flow of control, such as IF, ELSE, DO WHILE, and so on.

### Usage

Use ON EXIT MENU to execute a specified command when the user moves the cursor off certain pads in a specified menu. ON EXIT MENU associates the command only with pads that do not have an ON EXIT PAD command assigned to them. ON EXIT MENU, in effect, is the default command for every pad that does not have its own ON EXIT PAD command.

If you omit <command> when you use ON EXIT MENU, any command previously assigned for the specified menu is disabled.

**Example**

In the following example, when the user selects a menu pad, a box with explanatory information appears on the screen. The ON EXIT MENU commands runs a Cleanup procedure. As the user moves from the original pad to a new pad, this procedure removes the original pad's explanation box from the screen.

```

SET TALK OFF
DEFINE MENU Main
* Define pads...
DEFINE PAD pad1 OF Main AT 0,0 PROMPT "Plains States"
DEFINE PAD pad2 OF Main AT 0,15 PROMPT "Heartland"
DEFINE PAD pad3 OF Main AT 0,27 PROMPT "District of Columbia"
DEFINE PAD pad4 OF Main AT 0,50 PROMPT "Border States"
* Define pad actions...
ON EXIT MENU Main DO Cleanup
ON PAD pad1 OF Main DO Explain WITH "Iowa", "Kansas", "Minnesota", "Missouri",;
    "Nebraska", "North Dakota", "South Dakota"
ON PAD pad2 OF Main DO Explain WITH "Illinois", "Indiana", "Michigan",;
    "Ohio", "Wisconsin"
ACTIVATE MENU MAIN

* Procedures =====
PROCEDURE Explain
PARAMETERS state1, state2, state3, state4, state5, state6, state7
numstates = PCOUNT()
@ 10,1 TO 12+numstates, 45 DOUBLE
@ 11,3 SAY "This menu covers the following states:"
cntr = 1
DO WHILE cntr <= numstates
    z = "state"+LTRIM( STR( cntr ) )
    @ 11+cntr, 6 SAY &z
    cntr = cntr + 1
ENDDO
RETURN
*EOP Explain

PROCEDURE Cleanup
@ 10,1 CLEAR TO 19,45
RETURN
*EOP Cleanup

```

**See Also**

BAR(), DEFINE BAR, DEFINE MENU, DEFINE PAD, DEFINE POPUP, MENU(), ON BAR, ON EXIT BAR, ON EXIT PAD, ON EXIT POPUP, ON MENU, ON PAD, ON POPUP, ON SELECTION BAR, ON SELECTION MENU, ON SELECTION PAD, ON SELECTION POPUP, PAD(), POPUP(), PROMPT()

## ON EXIT PAD

ON EXIT PAD executes a specified command when the user moves the cursor off a specified pad in a menu.

### Syntax

ON EXIT PAD <pad name> OF <menu name> [<command>]



**NOTE** *<command> is any command except one that changes the flow of control, such as IF, ELSE, DO WHILE, and so on.*

### Usage

Use ON EXIT PAD to execute a specified command when the user moves the cursor off a specified pad in a menu. The user can move the cursor off a pad in the following ways:

- Use the ← and → keys to move the cursor to another pad.
- Click or double-click another pad.
- Drag the mouse cursor to another pad.
- Press the **Alt** key and the first letter of another pad's prompt.
- Press **Esc**.

To execute a command when the user *selects* (highlights) a pad, use ON PAD. To execute a command when the user *chooses* a pad, use ON SELECTION PAD.

To trigger the same command on *all* pads, use ON EXIT MENU instead of ON EXIT PAD.

If you omit <command> when you use ON EXIT PAD, any command previously assigned for the specified pad is disabled.

### Example

In the following example, when the user selects a menu pad, a box with explanatory information appears on the screen. The ON EXIT PAD commands run a Cleanup procedure. This procedure removes the previous pad's explanation box from the screen when the user leaves one pad to go to another.

## ON EXIT PAD ON EXIT POPUP

```
DEFINE MENU Main
* Define pads...
DEFINE PAD pad1 OF Main AT 0,0 PROMPT "Plains States"
DEFINE PAD pad2 OF Main AT 0,15 PROMPT "Heartland"
DEFINE PAD pad3 OF Main AT 0,27 PROMPT "District of Columbia"
DEFINE PAD pad4 OF Main AT 0,50 PROMPT "Border States"
* Define pad actions...
ON PAD pad1 OF Main DO Explain WITH "Iowa", "Kansas", "Minnesota", "Missouri",;
    "Nebraska", "North Dakota", "South Dakota"
ON EXIT PAD pad1 OF Main DO Cleanup
ACTIVATE MENU MAIN

* Procedures =====
PROCEDURE Explain
PARAMETERS state1, state2, state3, state4, state5, state6, state7
numstates = PCOUNT()
@ 10,1 TO 12+numstates, 45 DOUBLE
@ 11,3 SAY "This menu covers the following states:"
  cntr = 1
  DO WHILE cntr <= numstates
    z = "state"+LTRIM(STR(cntr))
    @ 11+cntr, 6 SAY &z
    cntr = cntr + 1
  ENDDO
RETURN
*EOP Explain

PROCEDURE Cleanup
@ 10,1 CLEAR TO 19,45
RETURN
*EOP Cleanup
```

### See Also

BAR(), DEFINE BAR, DEFINE MENU, DEFINE PAD, DEFINE POPUP, MENU(),  
ON BAR, ON EXIT BAR, ON EXIT MENU, ON EXIT POPUP, ON MENU,  
ON PAD, ON POPUP, ON SELECTION BAR, ON SELECTION MENU,  
ON SELECTION PAD, ON SELECTION POPUP, PAD(), POPUP(), PROMPT()

---

## ON EXIT POPUP

ON EXIT POPUP executes a specified command when the user moves the cursor off certain bars in a specified pop-up menu.

### Syntax

ON EXIT POPUP <popup name> [<command>]



**NOTE** *<command>* is any command except one that changes the flow of control, such as *IF*, *ELSE*, *DO WHILE*, and so on.

## Usage

Use **EXIT POPUP** to execute a specified command when the user moves the cursor off certain bars in a specified pop-up menu. **ON EXIT POPUP** associates the command only with bars that do not have an **ON EXIT BAR** command assigned to them. **ON EXIT POPUP**, in effect, is the default command for every bar that does not have its own **ON EXIT BAR** command.

If you omit *<command>* when you use **ON EXIT POPUP**, any command previously assigned for the specified pop-up menu is disabled.

To execute a command when the user *selects* (highlights) a pop-up menu, use **ON POPUP**. To execute a command when the user *chooses* a pop-up menu, use **ON SELECTION POPUP**.

## Example

In the following example, the **Scale** pop-up menu is defined to play a musical scale. The **ON BAR** commands execute the **Soundoff** procedure, the **ON SELECTION BAR** commands execute the **Frequency** procedure, and the single **ON EXIT POPUP** command executes the **Cleanup** procedure for all the bars.

```
MiddleC = 261.6
ratio 1
DEFINE POPUP Scale FROM 1,1
* Define bars...
DEFINE BAR 1 OF Scale PROMPT "Middle C"
DEFINE BAR 2 OF Scale PROMPT "B"
.
.
.
* Define bar actions...
ON EXIT POPUP Scale DO Cleanup
ON BAR 1 OF Scale DO Soundoff WITH 1
ON SELECTION BAR 1 OF Scale DO Frequency WITH 1
ON BAR 2 OF Scale DO Soundoff WITH 15/16
ON SELECTION BAR 2 OF Scale DO Frequency WITH 15/16
.
.
.
```

## See Also

**BAR()**, **DEFINE BAR**, **DEFINE MENU**, **DEFINE PAD**, **DEFINE POPUP**, **MENU()**, **ON BAR**, **ON EXIT BAR**, **ON EXIT MENU**, **ON EXIT PAD**, **ON MENU**, **ON PAD**, **ON POPUP**, **ON SELECTION BAR**, **ON SELECTION MENU**, **ON SELECTION PAD**, **ON SELECTION POPUP**, **PAD()**, **POPUP()**, **PROMPT()**

## ON MENU

ON MENU executes a specified command when the user selects (highlights) certain pads in a menu.

### Syntax

ON MENU <menu name> [<command>]



**NOTE** *<command> is any command except one that changes the flow of control, such as IF, ELSE, DO WHILE, and so on.*

### Usage

Use ON MENU to execute a command when the user selects certain pads in a specified menu. ON MENU associates the command only with pads that do not have an ON PAD command assigned to them. ON MENU, in effect, is the default ON PAD command for every pad that does not have its own ON PAD command.

If you omit <command> when you use ON MENU, the command previously assigned for the specified menu is disabled.

### Example

The following example uses ON MENU to display the names of types of files when the user moves the cursor over the pads. The ON PAD command for pad1 overrides that pad's ON MENU command.

```
DEFINE MENU Files
* Define pads
DEFINE PAD pad1 OF Files AT 0,0 PROMPT "Action"
DEFINE PAD pad2 OF Files AT 0,15 PROMPT "Database"
DEFINE PAD pad3 OF Files AT 0,30 PROMPT "Queries"
* Define pad actions
ON MENU Files DO Showfiles
ON PAD pad1 OF Files ACTIVATE POPUP Setact
.
.
.
```

### See Also

BAR(), DEFINE BAR, DEFINE MENU, DEFINE PAD, DEFINE POPUP, MENU(), ON BAR, ON EXIT BAR, ON EXIT MENU, ON EXIT PAD, ON EXIT POPUP, ON PAD, ON POPUP, ON SELECTION BAR, ON SELECTION MENU, ON SELECTION PAD, ON SELECTION POPUP, PAD(), POPUP(), PROMPT()

---

## ON MOUSE

ON MOUSE detects when the user clicks the left mouse button and executes a command when the button is released.

### Syntax

ON MOUSE [<command>]

### Usage

Use ON MOUSE to detect when the user clicks the left mouse button and then execute a command after the button is released.

ON MOUSE is active in the following areas:

- Outside of the @ GET regions during a READ or EDIT.
- Outside of all pads and pop-up menus, and on the left and right borders of active user-defined menus and pop-up menus.
- Outside of an active window. ON MOUSE is also active in the window's border unless the window contains a BROWSE screen.

ON MOUSE is inactive in the following areas and situations:

- In any screen where the internal dBASE IV menu system is active. A highlight in a pull-down menu indicates an active menu system.
- User-defined menu pads.
- On the BROWSE surface.
- On the dBASE design surfaces for data, forms, reports, and labels.
- In the Applications Generator.
- Inside @ GET regions.
- Inside user-defined menu pads.
- When SET MOUSE is off.
- When the user presses the left mouse button in an area unavailable to ON MOUSE, then moves and releases the left mouse button in an area sensitive to ON MOUSE.

### Example

The following example shows how you use ON MOUSE to execute a procedure when dBASE IV detects a mouse click.

## ON MOUSE ON PAD

```
ON MOUSE DO Chekmous
@ 2,5 TO 3,10          && Draw mouse box
DECLARE marray[1,5]
marray[1,1] = 2
marray[1,2] = 5
marray[1,3] = 3
marray[1,4] = 10
marray[1,5] = "First mouse box: 2,5 to 3,10"
Mpause = 'Click mouse in box'
@ 10,10 GET Mpause
READ

PROCEDURE Chekmous
mousrow = MROW()      && get the row position of the mouse
mouscol = MCOL()      && get the column position of the mouse
IF mousrow >= marray[1,1] .AND. mousrow <= marray[1,3]
  IF mouscol >= marray[1,2] .AND. mouscol <= marray[1,4]
    @ 10,10 SAY "You clicked the mouse on the box"
  ENDIF
ENDIF
RETURN
```

### See Also

ISMOUSE(), MCOL(), MROW(), SET(), SET MOUSE

---

## ON PAD

ON PAD executes a specified command when a menu pad is selected (highlighted).

### Syntax

```
ON PAD <pad name> OF <menu name> [<command>]
```

or

```
ON PAD <pad name> OF <menu name> [ACTIVATE POPUP <popup name>]
```



**NOTE** *<command> is any command except one that changes the flow of control, such as IF, ELSE, DO WHILE, and so on.*

### Usage

Use ON PAD to execute a specified command when the user selects (highlights) a particular menu pad. The user can select a pad in the following ways:



- Use the ← and → keys to move the cursor to the pad.
- Press the **ALT** key and the first letter of the pad's prompt.
- Click on the pad.
- Drag the mouse cursor to the pad.

Typically, **ACTIVATE POPUP** <popup name> is used with **ON PAD** to display a popup menu on the screen when the user selects a pad.

To execute a command when the user chooses (by pressing **Enter** or double-clicking the mouse) a pad, use **ON SELECTION PAD**. To execute a command when the user moves the cursor off a pad, use **ON EXIT PAD**.

To trigger the same command on *all* pads, use **ON MENU** instead of **ON PAD**.

If you omit <command> when you use **ON PAD**, the command previously assigned for the specified menu bar is disabled.

### Example

In the following example, **dBASE** lists all **.FRM** files when the user selects the **p2** pad.

```
DEFINE MENU Main
DEFINE PAD p1 OF Main PROMPT "Help"
DEFINE PAD p2 OF Main PROMPT "Show reports"
ON SELECTION PAD p1 OF Main DO Help
ON PAD p2 OF Main DIR *.frm
ACTIVATE MENU Main
```

### See Also

**BAR()**, **DEFINE BAR**, **DEFINE MENU**, **DEFINE PAD**, **DEFINE POPUP**, **MENU()**, **ON BAR**, **ON EXIT BAR**, **ON EXIT MENU**, **ON EXIT PAD**, **ON EXIT POPUP**, **ON MENU**, **ON PAD**, **ON POPUP**, **ON SELECTION BAR**, **ON SELECTION MENU**, **ON SELECTION PAD**, **ON SELECTION POPUP**, **PAD()**, **PADPROMPT()**, **POPUP()**, **PROMPT()**

---

## 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.

**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 POPUP

ON POPUP executes a specified command when the user selects (highlights) certain bars in a pop-up menu.

### Syntax

ON POPUP <popup name> [<command>]



**NOTE** *<command> is any command except one that changes the flow of control, such as IF, ELSE, DO WHILE, and so on.*

### Usage

Use ON POPUP to execute a command when the user selects certain bars in a specified pop-up menu. ON POPUP associates the command only with bars that do not have an ON BAR command assigned to them. ON POPUP, in effect, is the default command for every bar that does not have its own ON BAR command.

If you omit <command> when you use ON POPUP, any command previously assigned for the specified pop-up menu is disabled.

### Example

The following example uses ON POPUP to display help messages to the right of a pick list.

```
DEFINE POPUP filetype FROM 5,12
* Define bars
DEFINE BAR 1 OF filetype PROMPT ".DBF"
DEFINE BAR 2 OF filetype PROMPT ".QBE"
DEFINE BAR 3 OF filetype PROMPT ".SCR"
DEFINE BAR 4 OF filetype PROMPT ".FRM"
DEFINE BAR 5 OF filetype PROMPT ".LBL"
* Define bar actions
ON POPUP filetype DO Showhelp
ON EXIT POPUP filetype DO Closehelp
ACTIVATE POPUP filetype
RETURN
```

```
* Procedures =====
PROCEDURE Showhelp
DO CASE
  CASE PROMPT() = ".DBF"
    @ 7,27 SAY "These files store database records"

  CASE PROMPT() = ".QBE"
    @ 7,27 SAY "These files store query files"

  CASE PROMPT() = ".SCR"
    @ 7,27 SAY "These files store edit screen formats"

  CASE PROMPT() = ".FRM"
    @ 7,27 SAY "These files store reports"

  CASE PROMPT() = ".LBL"
    @ 7,27 SAY "These files store labels"
ENDCASE
RETURN

PROCEDURE Closehelp
@ 7,27 CLEAR TO 7,79
RETURN
```

### See Also

BAR(), BARCOUNT(), BARPROMPT(), DEFINE BAR, DEFINE MENU, DEFINE PAD, DEFINE POPUP MENU, MENU(), ON BAR, ON EXIT BAR, ON EXIT MENU, ON EXIT PAD, ON EXIT POPUP, ON MENU, ON PAD, ON SELECTION BAR, ON SELECTION MENU, ON SELECTION PAD, ON SELECTION POPUP, PAD(), PADPROMPT(), POPUP(), PROMPT()

---

## ON READERROR

Use ON READERROR for error trapping and recovery during full-screen operations.

### Syntax


ON READERROR [<command>]

## ON READERROR ON SELECTION BAR

### 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.

 **TIP** *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.*

### See Also

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

---

## ON SELECTION BAR

ON SELECTION BAR executes a specified command when the user chooses a bar in a pop-up menu either by pressing **Enter** when the cursor is on the bar, or by double-clicking the bar.

### Syntax

ON SELECTION BAR <expN> OF <popup name> [<command>]

 **NOTE** *<command> is any command except one that changes the flow of control, such as IF, ELSE, DO WHILE, and so on.*

### Usage

Use ON SELECTION BAR to execute a specified command when the user chooses a particular bar in a pop-up menu. The user can choose a bar in the following ways:

- Press **Enter**.
- Press the first letter of the bar's prompt.
- Click or double-click the bar.

To execute a command when the user *selects* (highlights) a bar, use ON BAR. To execute a command when the user moves the cursor off a bar, use ON EXIT BAR.

If you omit <command> when you use ON SELECTION BAR, the command previously assigned for the specified menu bar is disabled.

### Example

In the following example, dBASE lists all of the database files in the current directory when the user chooses bar 1.

```
DEFINE POPUP P1 FROM 1,1 MESSAGE "Files"  
DEFINE BAR 1 OF P1 PROMPT "Database Files"  
DEFINE BAR 2 OF P1 PROMPT "Query Files"  
ON SELECTION BAR 1 OF P1 DIR  
ON SELECTION BAR 2 OF P1 DIR *.qbe  
ACTIVATE POPUP P1
```

### See Also

BAR(), BARCOUNT(), BARPROMPT(), DEFINE BAR, DEFINE MENU, DEFINE PAD, DEFINE POPUP, MENU(), ON BAR, ON EXIT BAR, ON EXIT MENU, ON EXIT PAD, ON EXIT POPUP, ON MENU, ON PAD, ON POPUP, ON SELECTION MENU, ON SELECTION PAD, ON SELECTION POPUP, PAD(), POPUP(), PROMPT()

---

## ON SELECTION MENU

ON SELECTION MENU executes a specified command when the user chooses certain pads in a specified menu either by pressing **Enter** when the cursor is on the pad, or by double-clicking the pad.

### Syntax

ON SELECTION MENU <menu name> [<command>]



**NOTE** <command> is any command except one that changes the flow of control, such as *IF*, *ELSE*, *DO WHILE*, and so on.

### Usage

Use ON SELECTION MENU to execute a specified command when the user chooses certain pads in a specified menu. ON SELECTION MENU associates the command only with pads that do not have an ON SELECTION PAD command assigned to them. ON SELECTION MENU, in effect, is the default command for every pad that does not have its own ON SELECTION PAD command.

If you omit <command> when you use ON SELECTION MENU, any command previously assigned for the specified menu is disabled.

### Example

In the following example, only pad1 is assigned an ON SELECTION PAD command. The other pads use the ON SELECTION MENU command, executing the Showfiles procedure when they are chosen. Pad1 uses ON SELECTION PAD with no <command> to disable the pad from using the ON SELECTION MENU command. This pad doesn't need an ON SELECTION command since it triggers a pop-up menu when it is selected (highlighted).

```
DEFINE MENU Files
* Define pads...
DEFINE PAD pad1 OF Files AT 0,0 PROMPT "Action"
DEFINE PAD pad2 OF Files AT 0,15 PROMPT "Database"
DEFINE PAD pad3 OF Files AT 0,30 PROMPT "Queries"
DEFINE PAD pad4 OF Files AT 0,45 PROMPT "Forms"
DEFINE PAD pad5 OF Files AT 0,60 PROMPT "Reports"
* Define pad actions...
ON SELECTION MENU Files DO Showfiles
ON SELECTION PAD pad1 OF Files
ON PAD pad1 OF Files ACTIVATE POPUP Setact
.
.
.
ACTIVATE MENU Files
```

### See Also

BAR(), DEFINE BAR, DEFINE MENU, DEFINE PAD, DEFINE POPUP, MENU(), ON BAR, ON EXIT BAR, ON EXIT MENU, ON EXIT PAD, ON EXIT POPUP, ON MENU, ON PAD, ON POPUP, ON SELECTION BAR, ON SELECTION PAD, ON SELECTION POPUP, PAD(), POPUP(), PROMPT()

---

## ON SELECTION PAD

ON SELECTION PAD executes a specified command when the user chooses a pad in a menu either by pressing **Enter** when the cursor is on the pad, or by double-clicking the pad.

### Syntax

ON SELECTION PAD <pad name> OF <menu name> [<command>]



**NOTE** <command> is any command except one that changes the flow of control, such as IF, ELSE, DO WHILE, and so on.



## Usage

Use ON SELECTION PAD to execute a specified command when the user chooses a particular pad in a menu. The user can choose a pad in the following ways:

- Press **Enter** when the pad is highlighted.
- Click or double-click the pad.

To execute a command when the user *selects* (highlights) a pad, use ON PAD. To execute a command when the user moves the cursor off a PAD, use ON EXIT PAD.

To trigger the same command on *all* pads, use ON SELECTION MENU instead of ON SELECTION PAD.

If you omit <command> when you use ON SELECTION PAD, the command previously assigned for the specified menu pad is disabled.

## Example

In the following example, dBASE runs a Help program when the user chooses pad p1.

```
DEFINE MENU Main
DEFINE PAD p1 OF Main PROMPT "Help"
DEFINE PAD p2 OF Main PROMPT "Show reports"
ON SELECTION PAD p1 OF Main DO Help
ON PAD p2 OF Main DIR *.frm
ACTIVATE MENU Main
```

## See Also

BAR(), DEFINE BAR, DEFINE MENU, DEFINE PAD, DEFINE POPUP, MENU(), ON BAR, ON EXIT BAR, ON EXIT MENU, ON EXIT PAD, ON EXIT POPUP, ON MENU, ON PAD, ON POPUP, ON SELECTION BAR, ON SELECTION MENU, ON SELECTION POPUP, PAD(), POPUP(), PROMPT()

---

# ON SELECTION POPUP

ON SELECTION POPUP executes a specified command when the user chooses certain bars in a specified pop-up menu either by pressing **Enter** when the cursor is on the bar, or by double-clicking the bar.

## Syntax

ON SELECTION POPUP <popup name>/ALL [BLANK] [<command>]



**NOTE** <command> is any command except one that changes the flow of control, such as IF, ELSE, DO WHILE, and so on.

### Usage

Use ON SELECTION POPUP to execute a specified command when the user chooses certain bars in a specified pop-up menu. ON SELECTION POPUP associates the command only with bars that do not have an ON SELECTION BAR command assigned to them. ON SELECTION POPUP, in effect, is the default command for every bar that doesn't have its own ON SELECTION BAR command.

If you omit <command> when you use ON SELECTION POPUP, any command previously assigned for the specified pop-up menu is disabled.

If you use ALL, the command you specify applies to all the pop-up menus. BLANK clears the pop-up menu from the screen before executing the command. The pop-up menu is redrawn after the command is executed.

To execute a command when the user *selects* (highlights) a bar in a pop-up menu, use ON POPUP. To execute a command when the user moves the cursor off a pop-up menu, use ON EXIT POPUP.

### Example

The following example uses ON SELECTION POPUP to display the values of each field in a database file. The field values are displayed when the user chooses a field name from the list by pressing **Enter** or clicking the mouse.

```
DEFINE POPUP Pickfield FROM 0,5 PROMPT STRUCTURE
ON SELECTION POPUP Pickfield DO Showfield WITH PROMPT()
ACTIVE POPUP Pickfield
RETURN

* Procedures =====
PROCEDURE Showfield
PARAMETERS fieldname
CLEAR
DEFINE POPUP Showfield FROM 3,30 PROMPT FIELD $fieldname
ON SELECTION POPUP Showfield BLANK DO Editrecord ;
    WITH fieldname, PROMPT()
ACTIVATE POPUP Showfield
RETURN
*EOP Showfield

PROCEDURE Editrecord
PARAMETERS fieldname, fieldvalue
LOCATE FOR &fieldname = fieldvalue
EDIT
RETURN
*EOP Editrecord
```

**See Also**

BAR(), BARCOUNT(), BARPROMPT(), DEFINE BAR, DEFINE MENU, DEFINE PAD, DEFINE POPUP, MENU(), ON BAR, ON EXIT BAR, ON EXIT MENU, ON EXIT PAD, ON EXIT POPUP, ON MENU, ON PAD, ON POPUP, ON SELECTION BAR, ON SELECTION MENU, ON SELECTION PAD, PAD(), PADPROMPT(), 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. However if the deleted records contained memo fields, the corresponding .dbt file will not be reduced. In order to reduce the size of the memo file you must COPY the original database file. The corresponding .dbt memo file will then be copied and only contain the remaining memo field information.

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 Case**

In a multi-user 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.

If you pass more parameters than specified the excess parameters are ignored. If you pass less parameters than specified the excess variables are set to a logical **.F**. You can use the function **PCOUNT()** to find out how many parameters are actually passed.

You may pass up to 50 parameters from the calling program. There is a limit of 10 literals per **PROCEDURE** and nine per **FUNCTION**. User-defined **FUNCTIONs** reserve one space for the return value. You will get an error message if you exceed the number of literals.

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.

**PARAMETERS** treats **PUBLIC ARRAYS** and **PUBLIC** memory variables differently.

A **PUBLIC ARRAY** element passes its value to the subroutine, and the element itself remains unchanged by the actions of the called subroutines.

A **PUBLIC** memory variable passes a reference to its actual memory location to the subroutine, so the memory variable reflects any changes made by the called subroutines.

If you want an array element to reflect changes made by subroutines, store the value of the element to a **PUBLIC** memory variable and pass the memory variable to the subroutine. Upon return store the changed value of the memory variable to the array element.

### **See Also**

@, **DO**, **FUNCTION**, **PCOUNT()**, **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 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 within each macro library. 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 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 file specified by SYSPROC in Config.db.
2. Look in the current object file.
3. Look in the SET PROCEDURE file for the PROCEDURE filename, and locate the first instance of that PROCEDURE name.
4. Look in other open object files, in most-recently opened order.
5. Look for an object file with the procedure name, and open it.
6. Look in the SET LIBRARY file, if active.
7. Search for a .dbo file of that name.
8. Look for a program file with the procedure name, and compile it.
9. 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 are limited to a maximum of 963 procedures per file. Each procedure must begin with the keyword PROCEDURE. When a procedure begins with a PROCEDURE command, you cannot include an expression in the RETURN command line.

Procedure names are of unlimited length, but only the first nine characters are used. 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 Procl.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
```

To activate a procedure within Proc1.prg:

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

### See Also

COMPILE, DEBUG, DO, FUNCTION, PARAMETERS, RETRY, RETURN, SET LIBRARY, SET PROCEDURE

Chapter 15 of *Programming in dBASE IV*

---

## 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.

**M** **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 2-11 summarizes the database security types, how to implement each security type, and the results of security implementation.

Table 2-11 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 2-11.

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, dBASE IV looks for a Dbsystem.db file via the DOS path. 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

See Chapter 14, "Using the Tools Menu," in *Using dBASE IV* for more information about PROTECT. If you purchased the dBASE IV Network Access Pack, see Chapter 4, "Multi-user dBASE IV," in *Getting Started 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 list>

### Usage

PUBLIC memory variables or array elements are global variables that you can use in any program. Memory variables that you want to be global must be declared PUBLIC before they are given values. Specifying a memory variable from the dot prompt declares it to be PUBLIC. System memory variables are automatically declared public. The values of system variables may be changed by any program that uses them.

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 [WITH <expN>]
```

**Usage**

This is the only safe method for exiting from dBASE IV and returning to the operating system—any other method may damage open files and cause data loss. dBASE IV returns a 0 exit code if it terminates normally and a -1 code if it terminates abnormally. An abnormal termination can occur if the cache loader has difficulty loading or if your hard disk does not have sufficient free space for the dynamic overlay file.

**Options**

Using WITH, you can provide an integer from 0 to 255 that dBASE returns upon termination, based on the value of MOD ({return}, 256). You would use this feature if you had invoked your dBASE program from another program or batch file. Upon exiting the dBASE program the batch file or calling program would check for the exit code. A -1 appears as 255 to a batch file.

**See Also**

RUN, RUN()



## 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.

On multi-user systems, the database file must be in exclusive use before you can REINDEX .mdx files.

**Example**

```
. USE Transact
. REINDEX

Rebuilding index - C:\DBASE\TRANSACTION.MDX Tag: CLIENT_ID
100 % indexed      12 Records indexed
Rebuilding index - C:\DBASE\TRANSACTION.MDX Tag: ORD ER_ID
100 % indexed      12 Records indexed
```

**See Also**

BLANK, DESCENDING(), FOR(), INDEX, KEY(), MDX(), NDX(), PACK, REPLACE, SET EXCLUSIVE, SET INDEX, SET ORDER, SET UNIQUE, TAGNO(), 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 modules, menus, popups, screens, and windows from memory.

### Syntax

RELEASE <memvar list>

or

RELEASE ALL [LIKE/EXCEPT <skeleton>]

or

RELEASE MODULES [<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.

If you use RELEASE MODULES, MENUS, or any other keyword from this group without specifying the optional list, all occurrences of the particular object will be cleared from memory.

RELEASE MODULES 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 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. You cannot release the active window; either DEACTIVATE it, or ACTIVATE SCREEN or another window.

### 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 in the current directory, precede the filenames with the path.

### 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. Use COPY TO...WITH PRODUCTION to rename a production .mdx.

### 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:

```
. RENAME Letters.jim TO 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>] [REINDEX] [ 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.

### Options

ADDITIVE is used to build a memo field from several character strings. ADDITIVE is ignored unless the field is a memo field. When REPLACEing data in memo fields, the data type must be character or memo.

The <scope> option indicates the number of records to be affected according to the <scope> keyword:

- RECORD <n>
- NEXT <n>
- ALL
- NEXT

REINDEX rebuilds non-controlling indexes if the REPLACE command affected the index's key. REINDEX rebuilds these indexes after all replacements to the database file are completed. If you omit REINDEX, non-controlling indexes are updated after each record in the database file is REPLACED.

The Options section in the BLANK command entry of this chapter has a complete discussion of index updating, index rebuilding, and performance tuning with the REINDEX option.



**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, and 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 ↵.

### Special Case

In a multi-user environment, you can only use the REINDEX clause if the database is opened for exclusive use.

### 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

BLANK, REINDEX, REPLACE FROM ARRAY, 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>] [REINDEX] [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.

REINDEX specifies that all non-controlling affected indexes are rebuilt once the command is complete.

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.

## REPLACE FROM ARRAY REPORT FORM

```
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, BLANK, 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 dBASE IV*. 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 dBASE IV*.

If `SET ECHO` is `ON`, the source code of the `.frg` file is printed in addition to 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 (.prt) file 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 dBASE IV*

---

## 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 Case

In a multi-user 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 40
- The default alias underscore plus workarea number (`_<n>`).
- 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 40 (surrounded by parentheses if it is a simple variable)
- A character expression that yields a letter from A through J, an alias name, or an alias default of underscore plus number

### See Also

BEGIN TRANSACTION, END TRANSACTION, ISMARKED(), ROLLBACK, ROLLBACK(), SET EXCLUSIVE

---

## 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 with dBASE IV* for more information on memory variable default sizes and how you can change them in Config.db.

### Options

To retain the current memory variables, include the ADDITIVE option, which causes the RESTORED memory variables to be added to the current ones. Any current memory variables with the same name are overwritten.

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 SCREEN

RESTORE SCREEN 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 SCREENS, 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.



If **RETRY** re-executes the command successfully after the error-handling procedure corrects the error, program execution continues. You should, however, take provisional measures in case **RETRY** fails to re-execute the command successfully. For example, use a loop to return program control to the error-handling procedure so that the user can choose to retry or cancel the operation if the error message appears. Without this provision, program control would return to the command that caused the error, and attempt to execute the next line; however, because the previous error is unresolved, additional errors occur.

### 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 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 and value in a function.

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. This command works with data files. It is not intended to be used with program files to rollback programming changes.

**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.

**M 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 file `Myfile.mem` :

```
. SAVE TO Myfile
```

To **SAVE** all memory variables with names beginning with the letter *d* to the file `Myfile.mem`:

```
. SAVE ALL LIKE d* TO Myfile
```

To **SAVE** all memory variables, except those with names containing the letter *x* as the fourth character, to the file `Myfile.old`:

```
. 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.

## SAVE MACROS SAVE SCREEN

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, and .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
  @ 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 filename. 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/ENDSCAN

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.

Do not nest a SCAN command within another SCAN command.

### 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.

## SEEK SELECT

Use SEEK with SET NEAR OFF 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 40, or A through J. For work areas greater than 10, specify the work area by number or alias. You can open a database file in each work area you select.

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.

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. If the filename is not a valid alias, then dBASE assigns a default alias name beginning with the underscore character followed by the work area number.

Create a database file called x-x with a dummy field in it. From the dot prompt, enter:

```
. use x-x in 2
Default alias is _2
```

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
. SKIP Mskip           5
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 E).

SORT updates a catalog if one is in use and SET CATALOG is ON.

**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.

 **TIP** 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.

These commands:

```
. INDEX ON Lastname TO Lname
. COPY TO Namefile
```

perform an operation similar to

```
. SORT ON Lastname TO Namefile
```

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 rewriting 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.

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.

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.

 **TIP** *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.

The number of memory variables specified must be exactly the same as the number of fields SUMmed.

## SUM SUSPEND

The named array must be one-dimensional.

The number of array elements must be at least the same as the number of fields SUMmed.

If there are more array elements than needed, the extra variables or elements will not be filled and will contain their original values.

### 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 U at the **ACTION:** prompt in the debugger.

If SET TRAP is OFF and SET ESCAPE is ON, 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 ESCAPE, SET PROCEDURE, SET STEP, SET TALK, SET TRAP

## TEXT/ENDTEXT

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.

SET ALTERNATE TO a filename to create a file. Use SET ALTERNATE ON to open the file and receive a copy of the screen output.

### 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 ALTERNATE, SET PRINTER

---

## TOTAL

TOTAL sums the numeric fields of the active database file and creates a second database file to hold the results. The numeric fields in the TO database file contain the totals for all records that have the same key value in the original database.

### Syntax

```
TOTAL ON <key field> TO <filename> [FIELDS <fields list>]
[<scope>] [FOR <condition>] [WHILE <condition>]
```

### Defaults

The TO filename must include the path if it is not to be in the default directory: 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, 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. (To limit the fields in the TO file, use the SET FIELDS TO command.)

**TIP** *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

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 in the default directory, and SET PATH or SET DIRECTORY is not set to the file's location, you must precede the filename with its directory location.

### 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.

TYPE will display text after encountering a **Ctrl-Z** in a file except when **Ctrl-Z** is the first character in the 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

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] [REINDEX]
```

### 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.

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.

REINDEX rebuilds non-controlling indexes if the REPLACE command affected the index's key. REINDEX rebuilds these indexes after all replacements to the database file are completed. If you omit REINDEX, non-controlling indexes are updated after each record in the database file is REPLACED.

The Options section in the BLANK command entry of this chapter has a complete discussion of index updating, index rebuilding, and performance tuning with the REINDEX option.

In a multi-user environment, you can only use REINDEX if the database is opened for exclusive use.

**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**

BLANK, INDEX, REPLACE, SET EXCLUSIVE, 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] [NOLOG] [NOSAVE]
  [AGAIN]
```

## 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 activate the open catalog with 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 40 or from A to J. You can also use numeric expressions that yield 1 to 40 or character expressions that yield A to J. For work areas greater than 10, use numeric expressions.

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 file handle, but you can access up to 47 indexes. When you open an .ndx file, you use one 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 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 40, to refer to the work areas. For work areas greater than 10, specify the work area by number or alias. 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. 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.

NOLOG allows you to open and close files during a transaction without them becoming part of the transaction log file.

NOSAVE allows you to USE a .dbf as a temporary file. When you close a file opened with NOSAVE it is erased from the disk. If you inadvertently open a file with the NOSAVE option, use COPY TO with the PRODUCTION option to save all the data.

AGAIN allows you to open an already opened database file in another work area. Each image of the open file is a fully functioning version and it is up to the programmer to keep track of which version is the most recently updated.

### 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 the work area number preceded by an underscore.

If SET TALK is ON, no ALIAS option is specified, and the filename can't be used as the default alias, the work area number preceded by an underscore is used, and you see the message **Default alias is <\_n>**.

### 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), ALIAS(), BEGIN/END TRANSACTION, CATALOG, CLEAR ALL, CLOSE, COPY INDEX, COPY TAG, DBF(), DELETE TAG, INDEX, SELECT, SELECT(), SET CATALOG, SET EXCLUSIVE, SET INDEX, SET ORDER

---

**WAIT**

WAIT causes all dBASE IV processing to pause until any key is pressed.

**Syntax**

WAIT [<prompt>] [TO <memvar>]

**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 key you pressed to exit from the WAIT condition. The memory variable will contain a null value (ASCII value 0) if you press ↵. If you enter any other nonprinting character, the memory variable will contain the corresponding ASCII value.

**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 displays a confirmation box with YES and NO buttons.

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 Case

The database file must be in exclusive use before you issue the ZAP command.

### See Also

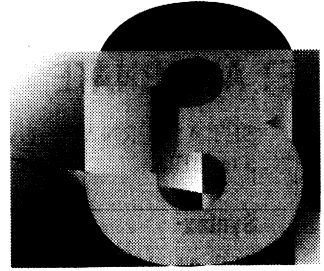
DELETE, PACK, SET SAFETY



# SET Commands



# 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 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 search path.

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 with dBASE IV* for more information on editing the Config.db file.

---

## SET ALTERNATE

SET ALTERNATE records all output other than that of full-screen commands to a text file.

### Syntax

```
SET ALTERNATE on/OFF
SET ALTERNATE TO [<filename> [ADDITIVE]]
```

### Default

The default for SET ALTERNATE is OFF.

The alternate file is written in the current directory, unless a path to another directory 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 CLOSE ALTERNATE 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()

---

## **SET BELL**

SET BELL controls the audible tone which alerts you to the end of a field, or to any erroneous or invalid entry.

### **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 with dBASE IV*.

To restore the default, type:

```
. SET BELL TO
```

### Examples

To test the bell frequency and duration from the dot prompt, type:

```
. SET BELL TO 20,1  
. ?CHR(7)
```

or:

```
. SET BELL TO 500,12  
. ?CHR(7)
```

### See Also

SET()

---

## SET BLOCKSIZE

SET BLOCKSIZE changes the default blocksize of memo fields and multiple index (.mdx) files by 512-byte increments.

### Syntax

```
SET BLOCKSIZE TO <expN>
```

### Usage

You may use a numeric argument from 1 to 63. This command changes the default blocksize of memo fields 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.

## SET BLOCKSIZE SET BORDER

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.

### See Also

COPY, COPY INDEXES, CREATE/MODIFY STRUCTURE, INDEX, REINDEX, REPLACE, 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 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.

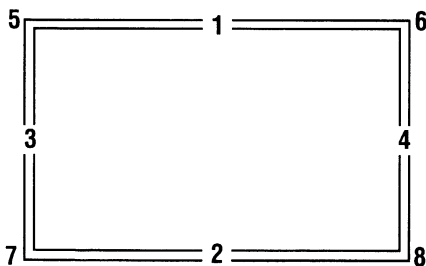


Figure 3-1 Order of border characters



**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, pop up, 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 PANEL gives an inverse video box as if ASCII 219 were entered. SET BORDER TO ↵ restores the default single line box.

The actual characters that form the box display on your screen depend upon which terminal definition file the current dBASE IV session is using. In general, an ASCII terminal will display a single-line box, even if you SET BORDER TO DOUBLE, but both single- and double-line boxes can display on the console.

If you supply an ASCII decimal value, as given in Appendix E, the corresponding ASCII character displays, unless the terminal definition file mapped the value to a different character.

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 with the plus sign at each corner, enter the command:

```
. SET BORDER TO , , , , "+" , "+" , "+" , "+"
```

**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 catalog files are in use. 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 its own buffer. If you wish to manipulate the catalog file you must first put it in use in a user accessible work area:

```
Catname = CATALOG()
USE (Catname) IN SELECT() AGAIN ALIAS Catalog
```

In order to update the catalog from within a program, temporarily turn it off:

```
Catname = CATALOG()

SaveCat = SET(" CATALOG" )
SET CATALOG OFF
USE (Catname) IN SELECT() ALIAS Catalog
*<update catalog>
USE IN Catalog
SET CATALOG TO (SaveCat)
```

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. The catalog does not show .mdx files because each .mdx file can contain up to 47 index keys. .ndx files contain only one key expression and are added to the 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**

Contains the full path name of the database if the file is not in the current directory.

**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
```

**Code**

When you have a catalog open, Code is the number dBASE IV assigns to each database file put into USE.

Program files are assigned 0. A database file is assigned a number when it is created. Each new database file gets the next higher code number. Files associated with a database file such as index, format, label, query, report form, screen, and view are assigned the same code number as the database file they reference.

**Tag**


This field is not used.

## Adding Entries

When SET CATALOG is ON, a new entry is added automatically to the active catalog when you use any of the following commands:

COPY STRUCTURE	IMPORT FROM
COPY STRUCTURE EXTENDED	INDEX
CREATE	JOIN
CREATE FROM	SET FILTER TO FILE
CREATE/MODIFY LABEL	SET FORMAT
CREATE/MODIFY QUERY	SET VIEW
CREATE/MODIFY REPORT	SORT
CREATE/MODIFY SCREEN	TOTAL
CREATE/MODIFY VIEW	USE

If the filename already exists in the catalog, you are prompted for a file title (if SET TITLE is ON). Each of these commands also allows you to select files using the ? query from the currently open catalog, regardless of whether SET CATALOG is ON or OFF.

 **NOTE** Use the SET TITLE OFF command to suppress the file title prompt if you do not want to enter file titles. To prevent the catalog from being updated, use the SET CATALOG OFF command.

## See Also

CATALOG(), SELECT(), SET(), SET TITLE, USE

---

## 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()  
02/14/91  
. SET CENTURY ON  
. ? DATE()  
02/14/1991  
. 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 68.

### 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, 68 from any other location.

The clock display is suppressed with full-screen menu commands if NOCLOCK=ON is in the Config.db file.

### See Also

SET()

---

## SET COLOR

SET COLOR allows the selection of colors and display attributes for use on color and monochrome monitors. Except for SET COLOR TO U and SET COLOR TO I, the colors you set are ignored on a monochrome monitor.

### 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.

The ISCOLOR() function returns a logical true (.T.) if SET COLOR is ON.

### 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 with dBASE IV* 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
```

## SET COLOR

Note that it is possible to change the foreground and the background to the same setting, making it impossible to see the cursor. *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	
<i>Symbol</i>	<i>Color</i>	<i>Symbol</i>	<i>Color</i>
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 BG+)	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 area

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) <sup>1</sup> 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 area (continued)

<b>Enhanced Area Groups</b>	<b>Displays Affected</b>
NORMAL	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 <sup>2</sup> Error messages in an error box <sup>2</sup> Cited command line in an error box <sup>3</sup> Navigation line messages <sup>2</sup> Available, unselected menu and list choices <sup>2</sup> Unavailable menu and list choices <sup>3</sup> File window contents <sup>2</sup> Help box interiors Bolded Help box text <sup>2</sup> Prompt box interiors Unselected prompt box buttons <sup>3</sup> Unselected error box buttons <sup>3</sup> Unselected Help box buttons <sup>3</sup>
TITLES	LIST/DISPLAY headings Help box headings Control Center banner <sup>2</sup> Control Center catalog name heading and filename Control Center filename and file description headings BROWSE field name headings BROWSE table grids <sup>3</sup> 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 <sup>3</sup> QBE file skeleton grids <sup>3</sup> QBE calculated fields skeleton grids <sup>3</sup> QBE condition box border <sup>3</sup>

(continued)

Table 3-4 Screen area (continued)

Enhanced Area Groups	Displays Affected Current field in the Applications
TITLES	QBE condition box heading Ruler line (in word wrap editor: MODIFY COMMAND, memo field editor) Text styles defined with the <b>Style</b> menu in layout editor <sup>2</sup> End-of-Report line in report design Static text in Applications Generator forms Underlined Help box text <sup>2</sup>
BOX	Menu borders File and sorted by information in <b>DOS Utilities</b> File window border in <b>DOS Utilities</b> and other lists Pick list borders Prompt box borders Label design layout box border Error box borders
HIGHLIGHT	Unselected Applications Generator object border Highlighted menu and list choices <sup>2</sup> 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 Selected button in Help box <sup>3</sup>
INFORMATION	Clock Help box borders Status line Selected button in error box Highlighted prompt box buttons <sup>3</sup>

(continued)

Table 3-4 Screen area (continued)

<b>Enhanced Area Groups</b>	<b>Displays Affected Current field in the Applications</b>
<b>FIELDS</b>	Prompt box data entry areas <sup>3</sup> Selected field in BROWSE <sup>3</sup> Editable fields in @...GET Selected fields in database design surface Unselected fields in the Applications Generator

<sup>1</sup> 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.

<sup>2</sup> 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.

<sup>3</sup> 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

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.

## SET COLOR SET CONFIRM

### Example

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 inverse video, type:

```
. SET COLOR OF INFORMATION TO I
```

### 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.

## SET CURRENCY SET CURRENCY LEFT/RIGHT

### 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 with dBASE IV*.

**Example**

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.

SET MARK changes the display format of MDY, DMY, and YMD.

You can also change the default date format by placing this option in the Config.db file. See *Getting Started with dBASE IV* 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 MARK

---

## 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 commands, 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.

## SET DECIMALS SET DEFAULT

### 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 PRECISION

---

## 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 *Getting Started with dBASE IV* 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.

## SET DELETED SET DELIMITERS

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(), RECALL, 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. SET DESIGN is local to the sub-procedure in which it is toggled. If a UDF or a procedure run from inside another procedure or program turns it OFF, SET DESIGN would be ON upon return to the calling procedure.

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

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

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 .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. The printer, in this case, is the one defined by the last SET PRINTER command, or the default print spooler if no SET PRINTER command was given.

@...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 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 start-up 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 <drive:>, RUN CD <path>, 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 with dBASE IV* to find out how to set the working drive and directory in the Config.db file.

### 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"
C:\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 25-, 43- and 50-line screens.

### Syntax

```
SET DISPLAY TO MONO/COLOR/EGA25/EGA43/MONO43/VGA25/
VGA43/VGA50
```

## SET DISPLAY SET ECHO

### Usage

You can use this command only if you have hardware that is capable of supporting monochrome and color display and an EGA, VGA, or equivalent graphics card. 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 PRINTER, 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 **Esc**. It lets you stop screen scrolling by pressing **Ctrl-S**, and to record these 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 interruptions. SET ESCAPE OFF also disables the operation of **Ctrl-S** 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.

**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.

For optimum performance, SET EXCLUSIVE ON whenever you are not planning to share a file.

### 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.

## 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
BLANK
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  $\downarrow$  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 FILE ? shows a list of available dBASE III PLUS filter (.qry) files.

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 except memo; 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 condition. Continuing with the environment of the preceding example:

```
. GOTO 3
TRANSACTION: 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 or 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** *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 in the current directory.

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, EDIT, INSERT, CREATE/MODIFY SCREEN, CLOSE, READ, SET(), SET DEVICE

---

## SET FULLPATH

SET FULLPATH permits or suppresses the display of a full path name with the DBF(), CATALOG(), MDX(), NDX(), 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 path name, whereas the same functions return only the drive and the filename in dBASE III PLUS.

Use this command in programs that use the DBF(), CATALOG(), MDX(), NDX(), 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 path name display.

**See Also**

DBF(), CATALOG(), MDX(), NDX(), SET()

## 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

<expN> is a range between 2 and 29.

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 with dBASE IV*.

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

Table 3-5 Default function key assignments

Key	Value	Key	Value
F1	RESERVED	F6	DISPLAY STATUS;
F2	ASSIST;	F7	DISPLAY MEMORY;
F3	LIST;	F8	DISPLAY;
F4	DIR;	F9	APPEND;
F5	DISPLAY STRUCTURE;	F10	EDIT;

If you enter SET FUNCTION...TO without a final expression, the function key will return to the standard assignments.


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 a keyboard with function keys beyond **F10**, 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 quotation marks 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.

## SET HEADINGS SET HELP

### 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.

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 with dBASE IV*.

### Examples

At 8:00 p.m., the clock display changes as follows:

```
. SET HOURS TO 12  
8:00:00 pm
```

```
. SET HOURS TO 24  
20:00:00
```

To return to the default:

```
. SET HOURS TO ↵
```

### See Also

SET(), SET CLOCK



## SET IBLOCK

SET IBLOCK lets you change the default size of the indexing block (also called *node*) that dBASE allocates to new index data (.MDX) files.

### Syntax

SET IBLOCK TO <expN>

### Default

The default value for SET IBLOCK is 1. Each unit is equivalent to 512 bytes. Index blocks, however, must be at least 1024 bytes; therefore, the default size of the index block is 1024 bytes.

### Usage

Use SET IBLOCK to improve performance by specifying the size of the index block that dBASE allocates to new .MDX files. Valid unit values for SET IBLOCK range from 1 to 63. Each unit except the first one (which is internally read as 2 and defaults to the minimum of 1024 bytes) equals 512 bytes. Therefore, .MDX block sizes can range from 1024 bytes to approximately 32K. For example, specifying SET IBLOCK TO 4 allocates 4 \* 512, or 2048 bytes to each block used in new .MDX files.

The .MDX indexes are composed of indexing blocks or nodes. Leaf nodes contain key values that refer to individual records and provide the relevant record for each key value. A leaf node contains information like the following:

Record Number	Key Value
23	Aberdeen
456	Abilene
1	Abington Township
...	...
234	Amsterdam

Since the IBLOCK setting determines the size of the nodes, the setting also determines the number of key values that can fit into each node. When a single node can't contain all of the key values in an index, dBASE creates one or more parent nodes. These "intermediate" nodes also contain key values. Instead of pointing to record numbers, however, intermediate nodes point to leaf nodes or other lower-level intermediate nodes.

For example, in the index of cities shown in the previous example, the last city in the leaf node is Amsterdam. The parent node for this index of cities might look like the following:

Block Number	Key Value
3	Amsterdam
4	Auburn
5	Bell Gardens
...	...

If you increase the size of the index block or node, and create a new .MDX file from this data with the `COPY TO...WITH PRODUCTION` command, the new and larger leaf nodes contain more key values and the city names in the parent node are further apart.

Whether you can improve performance by storing key values in larger or smaller nodes depends on several factors: the distribution of data, if database files will be linked together, the length of the key values, the value of `INDEXBYTES`, and the type of operation requested. Typically, every .MDX file contains more than one index tag. Finding the best setting for a given .MDX file requires trial and error experimentation because the best size for one index tag might not be the best size for another.

The following is a list of basic principles governing index performance:

- Since nodes may not be sequential, dBASE reads only one node at a time from the disk. Reading more than one node is usually inefficient because more than likely the second node is not the next node in the sequential list.
- Once a node is read into memory, dBASE attempts to store it there for later use. The amount of space devoted to caching the index nodes is determined by the setting of `INDEXBYTES`.
- When users link several database files together (e.g., with `SET RELATION`), performance is better if all the relevant nodes for the files are in memory simultaneously. For example, if a large node for file B pushes out the previously read node for file A, dBASE must find and read file A's node again from the user's disk when the node for file A needs to be used again. If both nodes remain in memory, performance will probably improve.
- When user's have many identical key values, dBASE might have to store them in many nodes. In this situation, performance might be improved by increasing the node size so that dBASE IV reads fewer nodes from the user's disk to load the same number of key values into memory.
- Small node sizes can cause performance degradation. This occurs because as nodes are read in and out, dBASE IV attempts to cache them all. When the small nodes are removed from memory by more recently read nodes, they leave unused spaces in memory that are too small to contain larger nodes. Over time, memory can become "fragmented," resulting in slower performance.

### Example

The following example shows how you use SET IBLOCK to change the node size:

```
USE Employee
SET IBLOCK TO 12
COPY TO Temp WITH PRODUCTION      && Uses the new IBLOCK value
USE Temp
SET SAFETY OFF
COPY TO Employee WITH PRODUCTION
SET SAFETY ON
USE
ERASE Temp.bdf
ERASE Temp.mdx
RETURN
```

### See Also

SET(), SET BLOCKSIZE, SET MBLOCK

---

## 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 ?

SET INDEX TO <filename list> [ORDER <.ndx filename>]

SET INDEX TO <filename list> [ORDER <.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 C1nt2  
. SET INDEX TO Cus_name, C1nt2 ORDER Fullname
```

## See Also

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

---

# SET INSTRUCT

SET INSTRUCT enables the display of prompt boxes or dBASE code while 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**

CREATE/MODIFY LABEL, CREATE/MODIFY REPORT, CREATE/MODIFY SCREEN, SET()

---

## **SET INTENSITY**

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

SET KEY allows the display of only those records of a database file whose ordering index key meets a specified range of conditions.

### Syntax

```
SET KEY TO [ <exp:match>/RANGE <exp:range>][IN<exp:WA>]
```

```
where <exp:range>::<exp:low>,<exp:high>
      <exp:low>[,]
      , <exp:high>
```

### Defaults

SET KEY TO turns off the current specified range.

### Usage

SET KEY requires that a database be in USE with an INDEX.

The data type of <exp:match>, <exp:low>, and <exp:high> must be the same as the data type of the controlling index key.

SET KEY looks for matches within the controlling index key according to the specified criteria.


SET KEY TO <exp:match> searches for an exact match.

SET KEY TO RANGE <exp:match> searches the index for values equal to or greater than <exp:match>.

SET KEY TO RANGE <exp:low>, is the same as SET KEY TO RANGE <exp:match>.

SET KEY TO RANGE ,<exp:high> searches the index for values equal to or less than <exp:high>.

SET KEY TO RANGE <exp:low>,<exp:high> searches the index for all values equal to or greater than <exp:low> and all values equal to or less than <exp:high>.

 **NOTE** *The range established by SET KEY takes immediate effect. To change the range you must reissue the command with new parameters.*

The SET KEY command does not affect the operation of record updating or the index key itself.

If the controlling index is discarded, the range determined by SET KEY is lost.

In selecting records for display, SET KEY has priority over SET FILTER. SET FILTER only works on the set of records that meet the criteria of SET KEY.

### **Examples**

```
. USE PEOPLE  
. INDEX ON Zip TAG Zip  
. SET KEY TO RANGE '60000','90000'  
. BROWSE
```

### **See Also**

INDEX, KEY(), MDX(), NDX(), TAG(), SET FILTER

---

## **SET LDCHECK**

SET LDCHECK lets you enable or disable language driver ID checking.

### **Syntax**

SET LDCHECK ON/off

### **Default**

The default value for SET LDCHECK is ON.


### **Usage**

Use SET LDCHECK to disable or enable dBASE's capability to check for language driver compatibility. This language driver ID checking functionality is important if you work with dBASE files created with different dBASE configurations or different international versions of dBASE because it warns you of conflicting language drivers.

A language driver contains information about the DOS code page (character set) and language tables that dBASE uses. The DOS code page, which you load through DOS, lets you type, display, and print characters for a particular language. The language tables, provided by dBASE, determine how dBASE alphabetizes and sorts data, handles extended ASCII (>127) characters, and converts characters from uppercase to lowercase and vice versa.

Since language drivers determine the character set and sorting rules that dBASE uses, a file created with the German version of dBASE, for example, is likely to contain extended ASCII characters that the US version doesn't use and, therefore, affects the way dBASE sorts the data. If you create a dBASE file with one language driver and then use that file with a different language driver, some of the extended ASCII characters will appear incorrectly and you will probably get incorrect results when querying data.



 **NOTE** *Using the same DOS code page doesn't guarantee language driver or data compatibility. Users can, for example, change the settings of the language table through two configuration settings: LangTable and AsciiSort. If two systems are using code page 437, but user A sets LangTables to ON while user B sets LangTables to OFF, dBASE uses different language drivers. For more information about the LangTable and AsciiSort settings, see the "Configuring dBASE IV" chapter in Getting Started.*

If you try to use a file created with a different language driver and SET LDCHECK is on, dBASE warns you of mismatched language drivers and gives you the option of continuing the operation with the current language driver or cancelling it.

You can use SET LDCHECK in your ON ERROR routines or in the same way you would use SET SAFETY for verification in your programs, as shown in the following example.

### Example

```

ON ERROR DO err_proc           && Execute err_proc when an error occurs
SET LDCHECK ON                 && Turn language driver checking on
USE Cities
ON ERROR                       && Turn error trapping off
RETURN

PROCEDURE err_proc
DO CASE
CASE ERROR() = 531             && Trap error for mismatched language driver
                                && for .MDX file
SET LDCHECK OFF                && Turn language driver checking off before
                                && taking recovery action

USE Cities
REINDEX                       && Reindex file so that it uses current
                                && language driver

SET LDCHECK ON
ENDCASE
RETURN

```

### See also

SET()

---

## SET LIBRARY

SET LIBRARY enables you to have a semi-permanent collection of procedures and functions to augment the built-in commands of dBASE IV.

**Syntax**

SET LIBRARY TO [<filename>]

**Usage**

SET LIBRARY is a complement to the SET PROCEDURE command and the SYSPROC setting of the Config.db file. The file described by <filename> is established as a set of library procedures and is part of the following calling order for procedures and functions:

1. Look in the SYSPROC = <filename>, if active.
2. Look in the currently executed .dbo file.
3. Look in the SET PROCEDURE TO <filename>, if active.
4. Look in the other .dbo files in the call chain, in most recently opened order.
5. Look in the SET LIBRARY TO <filename>, if active.
6. Search the disk for a .dbo file.
7. Search the disk for a .prg file.
8. Search for a .prs file.

By the judicious use of this search order you can use multiple procedures with the same name to perform different tasks.

SYSPROC is intended for procedures that take precedence over everything except internal dBASE commands.

SET PROCEDURE is intended to be application specific for procedures that replace your general procedures in SET LIBRARY.

SET LIBRARY are the generalized procedures used from application to application.

**See Also**

SET PROCEDURE

*Programming in dBASE IV* and the Procedures section in Chapter 1 of this book.

---

## **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. Table 3-6 lists the commands that automatically LOCK files.

When command execution is complete, the file or record is unlocked.

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.

Table 3-6 Commands that automatically LOCK files

<b>dBASE Command</b>	<b>Action</b>	<b>Level</b>	<b>Does SET LOCK OFF Disable LOCK?</b>
@GET/READ	edit	Record	No
APPEND FROM	update	File	No
APPEND [BLANK]	update	Record	No
AVERAGE	read-only	File	Yes
BROWSE	edit	Record	No

*(continued)*

Table 3-6 Commands that automatically LOCK files (continued)

<b>dBASE Command</b>	<b>Action</b>	<b>Level</b>	<b>Does SET LOCK OFF Disable LOCK?</b>
CALCULATE	read-only	File	Yes
CHANGE/EDIT	edit	Record	No
COPY TAG/INDEX	read/write	File	Yes on read; No on write
COPY [STRUCTURE]	read/write	File	Yes on read; No on write
COUNT	read-only	File	Yes
DELETE/RECALL	update	Record	No
DELETE/RECALL	update	File	No
EDIT	update	Record	No
INDEX	read/write	File	Yes on read; No on write
JOIN	read/write	File	Yes on read; No on write
LABEL	read-only	File	Yes
REPLACE	update	Record	No
REPLACE [<scope>]	update	File	No
REPORT	read-only	File	Yes
SET CATALOG ON	catalog	File	No
SORT	read/write	File	Yes on read; No on write
SUM	read-only	File	Yes
TOTAL	read/write	File	Yes on read; No on write
UPDATE	update	File	No

**See Also**

FLOCK(), SET()

*Getting Started 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 with dBASE IV*. If `_wrap` is true and streaming output is produced the effect of SET MARGIN is added to the effect of `_lmargin`. When `_wrap` is false, `_lmargin` and SET MARGIN are not additive.

### Examples

To set the left margin 10 spaces to the right of the first possible print position:

```
. SET MARGIN TO 10
. ? _ploffset
10
```

The margin setting can be changed with `_ploffset`:

```
. _ploffset=5
5
```

## SET MARGIN SET MARK

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()`, `_ploffset`, `_lmargin`, `_wrap`

---

## 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 quotation marks* 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 MBLOCK

SET MBLOCK lets you change the default size of the block that dBASE allocates to new memo field files (.DBT).

### Syntax

SET MBLOCK TO <expN>

### Default

The default value for SET MBLOCK is 8. Each unit is equivalent to 64 bytes; therefore, the default size of the memo block is 512 bytes.

### Usage

Use SET MBLOCK to improve the use of disk space by specifying the size of the block that dBASE allocates to new .DBT files. Valid unit values for SET MBLOCK range from 1 to 511. Each unit is equal to 64 bytes; therefore, .DBT block sizes range from 64 bytes to approximately 32K. For example, specifying SET MBLOCK TO 4 allocates 4 \* 64, or 256 bytes to each block used in new .DBT files.

dBASE stores data in each memo field in a group made up of as many blocks as needed. When the block sizes are large and the memo contents small, .DBT files contain unused space and become larger than necessary.

If you expect the contents of the memo fields to occupy less than 512 bytes (the default size allocated), set the block size to a smaller size to reduce wasted space. If you expect to store larger pieces of information in memo fields, increase the size of the block.

SET MBLOCK is similar to the older SET BLOCKSIZE command except for two advantages:

1. You can allocate different block sizes for memo field data (.DBT) and index data (.MDX), whereas SET BLOCKSIZE requires the same block size for both. To allocate block sizes for index data, use SET IBLOCK.
2. You can specify smaller blocks with SET MBLOCK than with SET BLOCKSIZE. SET BLOCKSIZE creates blocks in increments of 512 bytes, compared to 64 bytes with SET MBLOCK.

### Example

The following example shows how you can use SET MBLOCK to optimize existing .DBT files:

## SET MBLOCK SET MEMOWIDTH

```
USE Mydata                                && Mydata.dbt has 512-byte blocks
SET MBLOCK TO 1                            && Specify 64-byte blocks
* Create a new .DBT with new block sizes
COPY TO Temp WITH PRODUCTION
USE Temp                                    && Use new files
* Make a set of files with the original filename
SET SAFETY OFF
COPY TO Mydata WITH PRODUCTION
SET SAFETY ON
USE Mydata
ERASE Temp.dbf                             && Erase temporary copy
ERASE Temp.dbt
ERASE Temp.mdx
RETURN
```

### See Also

SET(), SET BLOCKSIZE, SET IBLOCK

---

## 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 with dBASE IV*.

### Usage

Use SET MEMOWIDTH TO to alter the column width of memo fields during output. If the system memory variable `_wrap` is set to true (.T.), the system variables `_lmargin` and `_rmargin` determine the memo width.

The @V (vertical stretch) picture function causes memo fields to be displayed in a vertical column when `_wrap` is true. When @V is equal to zero, memo fields word wrap within the SET MEMOWIDTH width. When @V is specified, the `_pcolno` system memory variable is incremented by the @V value. This allows you to change the appearance of the printed output of ??? commands by using the @V function.

### See Also

??? (@V FUNCTION), MEMLINES(), MLINE(), SET(), `_lmargin`, `_rmargin`, `_wrap`



## 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 except the ERROR clause of an @...GET always displays on the message line at the bottom of the screen.

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.

 **TIP** 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 and suppress the clock display before using line 0 for messages.*

### See Also

SET CLOCK, SET SCOREBOARD, SET STATUS

---

## SET MOUSE

SET MOUSE enables or disables the mouse cursor, if one is installed.

### Syntax

SET MOUSE ON/off

### Default

If a mouse driver is loaded, the default for SET MOUSE is ON.

### Usage

Use SET MOUSE OFF to remove the mouse cursor from the screen and ignore all subsequent mouse actions. To enable the mouse again, use SET MOUSE ON.

When you SET MOUSE ON, dBASE determines if a mouse driver is present and a mouse connected. If a mouse driver is missing, an error message is displayed.

### Example

The following example checks if there is a working mouse, stores the old mouse setting, and enables the mouse.

```
IF ISMOUSE()  
    oldmset = SET("MOUSE")  
    SET MOUSE ON  
ENDIF
```

### See Also

ISMOUSE(), MCOL(), MROW(), ON MOUSE, SET()

---

## 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 1, 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.

When SET TALK is ON, you are encouraged to SET ODOMETER to the highest value. The larger the value, the fewer screen updates will occur and the more performance will be improved.

**See Also**

SET(), SET TALK

---

## 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 <.ndx filename> / [TAG] <.mdx tagname>  
    [OF <.mdx filename>][NOSAVE]
```

**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.

### Options

NOSAVE instructs dBASE IV to delete any temporary index tags created by QBE or by the user. This option will delete the tag when the .mdx file is closed, and is helpful when a temporary index tag is needed.

### 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
```

## SET ORDER SET PATH

To set the master index back to the TAG Client:

```
. SET ORDER TO Client  
Master index: CLIENT
```

### See Also

DESCENDING(), DISPLAY, FOR(), INDEX, KEY(), MDX(), NDX(), ORDER(), REINDEX, SET INDEX, TAG(), TAGCOUNT(), TAGNO(), UNIQUE, 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 254 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
```

**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 full screen.

**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."

---

## SET POINT

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. 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.

Only one character can be used as point. If you define more than one character, dBASE uses only the first one.

### 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 20 digits.

Operations involving both type N and type F numbers will yield a type F result. Type F numbers have a precision of 15, regardless of the SET PRECISION value.

**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 device PRN.

SET PRINTER TO \\<computer name>\<printer name> = <destination>

SET PRINTER TO \\SPOOLER

SET PRINTER TO \\CAPTURE

SET PRINTER TO FILE <filename>

**Usage**

The first five forms of this command are used to direct or redirect printer output to different print devices. The sixth syntax produces printer-formatted files.

## Redirecting Screen Output to Printer

When SET PRINTER is ON, all screen output, including TALK, LIST, and ? is routed to the printer.

Use SET DEVICE TO PRINTER to direct @...SAY commands to the printer. SET PRINTER ON does not affect @...SAY commands.

## Printing to a Local Device

Use SET PRINTER TO <DOS device> to send printed output to a local printer. The default is the DOS print 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.

### 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 embedded 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.

## SET PRINTER SET PROCEDURE

Assuming the printer formatted file, `Filename.prt`, is on a disk 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 embedded 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 default printer.

### See Also

`???`, `???`, LABEL FORM, LIST/DISPLAY, PRINTJOB/END PRINTJOB, PRINTSTATUS(), 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 in the default directory 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 or a `.prs` 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 maximum of 963 procedures (routines). 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. Procedures may also be in the files specified by SYSPROC and SET LIBRARY TO.

### 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
```

## SET PROCEDURE SET REFRESH

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 LIBRARY

---

## 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

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.

**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>  
[, <expression> INTO <alias>...]]
```

**Default**

SET RELATION TO .J, 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.

 **TIP** You can save a working environment, including relations, with *CREATE VIEW <.vue filename> FROM ENVIRONMENT*, which creates a .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

## SET RELATION

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 VIEW FROM ENVIRONMENT, SET(), SET FIELDS, SET FILTER, SET SKIP, SET VIEW

---

## SET REPROCESS

SET REPROCESS sets the number of times dBASE IV tries a 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 to change the command execution process. A record or a file may be in use by another user, 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

---

## **SET SCOREBOARD**

SET SCOREBOARD displays the keyboard indicators and record delete status. In a multi-user environment it also displays the lock status.

### **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 multi-user environment, **RecLock** and **FileLock** indicate a record or file lock. If you SET SCOREBOARD OFF from within a program the current screen will be cleared.

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.

**Programming Note**

Toggling SET SCOREBOARD within a procedure clears the screen.

**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 with dBASE IV*.

**Example**

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
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 repeated record numbers are matches from related files.

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 with dBASE IV* to change your Config.db file.

---

## SET STATUS

SET STATUS determines whether the status bar displays at the bottom of the screen when you are working from the dot prompt, from within a program, and in full-screen editing commands such as APPEND, 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 system default Config.db file which is loaded during installation of dBASE IV. Your own Config.db file may also be used to override the default.

### 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 on line 0 of the screen. If both SET STATUS and SET SCOREBOARD are OFF, the scoreboard is not displayed.

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 in a format file below line 21. 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 OFF.

**Usage**

SET TALK ON 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.

SET TALK OFF reduces the number of characters that are sent to the screen and improves performance.

**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.

When you create files from the Control Center, and a catalog is set, these files are added to the catalog but you are not prompted to enter a Description.

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. You can open and modify a catalog file in any work area. But you cannot modify a catalog opened by **SET CATALOG TO**.

To modify a catalog:

```
. SET CATALOG TO  
. USE CATALOG IN SELECT()  
. SELECT <CATWA>  
. EDIT
```

### 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.

 **TIP** *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.*

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.

### Example

To LIST all of the Part\_names in the Items database file without repeating a part:

```
. USE Stock
. SET UNIQUE ON
. INDEX ON Part_name TO Itemname
100% indexed          10 Records indexed
. LIST Part_name
```

Record#	Part_name
12	BOOK CASE
4	CHAIR, DESK
11	CHAIR, SIDE
15	DESK, EXECUTIVE 5-FOOT
8	FILE CABINET, 2 DRAWER
9	FILE CABINET, 4 DRAWER
7	LAMP, FLOOR
1	SOFA, 6-FOOT
3	SOFA, 8-FOOT
6	TABLE, END

### See Also

FIND, INDEX, REINDEX, SEEK, SEEK(), SET(), 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, 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 from one to ten of each
- All relations between the database files
- The currently selected work area number
- The active field list
- The active filter
- The SET SKIP value
- The SET KEY range

### **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 for memo fields 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



# Functions



# 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 run time. 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 multi-user 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.

## ACCESS()

```
ACCEPT "Enter choice " TO Choice
DO CASE
  CASE Choice="1"
    IF ACCESS() < 3
      DO Program
    ELSE
      ? CHR(7)
      ? "Unauthorized ACCESS level"
      WAIT
      RETURN TO MASTER
    ENDIF
  CASE Choice="2"
    *
    *
    *
ENDCASE
```

### Special Cases

If dBASE IV does not find the Dbssystem.db file during start-up, 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, 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 system 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  $\pi$ . The SET DECIMALS command determines the number of decimal points 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 = P12
      12
. Mcos = x / y
      0.7071
. angle = ACOS(Mcos)
      0.7854
```

### See Also

ASIN(), ATAN(), ATN2(), COS(), DTOR(), FIXED(), FLOAT(), PI(), RTOD(), SET DECIMALS, SIN(), TAN()

---

## ALIAS()

ALIAS() returns the alias name of a specified work area. If you do not specify a work area, it returns the name of the current work area.

### Syntax

ALIAS([<alias>])

### Usage

The ALIAS() function returns the filename or unique alias associated with any work area. The optional alias parameter identifies a work area and can be a numeric expression from 1 to 40, a character expression from A through J, or a work area name. A work area name can be either a database filename or an alias name specified with the USE command.

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 <alias>, it is truncated to 7.

### Example

Check the alias name assigned to work area 20:

```
. USE Client ALIAS Customers IN 20 ORDER Lastname  
. ? ALIAS(20)  
CUSTOMERS  
. ? ALIAS("Customers")  
CUSTOMERS
```

### See Also

DBF(), SELECT(), USE

---

## ASC()

The ASC() function returns the ASCII decimal code of the first character from a character expression.

### Syntax

ASC(<expC>)

### Usage

Appendix E lists the decimal ASCII values for the 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  $-/2$  and  $+/2$ . The SET DECIMALS command determines the number of decimal points 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, SIN(), TAN()

---

## AT()

AT() returns a number that shows the starting position of a character string (substring) within another string or memo field. AT() starts the search from the *first* character of the string or memo field being searched.

### Syntax

AT(<expC1>,<expC2>[,<expN>])

or

AT(<expC1>,<memo field name>[,<expN>])

### Usage

Use the AT() function to find the starting position of a character string within another string or memo field. The search is case-sensitive and starts, one character at a time, from the first character of the string or memo field being searched.

You must specify the substring (<expC1>) you want to find, and the string or memo field (<expC2>) you want to search. You have the option of specifying the *n*th occurrence of the substring (<expN>) within the source string. If you omit this argument, AT() returns the starting position of the *first* occurrence of the substring.

If the substring is not found, or is longer than the source string or memo field, AT() returns a value of zero.



**NOTE** *If you are searching a memo field, AT() limits its search to approximately 64K of data.*

### Examples

```
. ? AT("ss", "Mississippi")
3
. ? AT("ss", "Mississippi", 2)
6
```

### See Also

LEFT(), RAT(), RIGHT(), STUFF(), 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 + /2 and - /2. The returned value is always a number that represents an angle size in radians. The SET DECIMALS command determines the number of decimal points 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, 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 + and - .

### 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 + and -. 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()

---

## BARCOUNT()

BARCOUNT() returns the number of bars in the active or specified pop-up menu.

### Syntax

BARCOUNT ([<expC>])

### Usage

Use this function to get the number of bars in a specified pop-up menu. This information is especially useful when the pop-up is defined with PROMPT FIELD, PROMPT FILES, or PROMPT STRUCTURE, and you need to know how many elements there are.

If you do not provide a pop-up name, BARCOUNT() returns the number of bars in the active pop-up menu. If none of the pop-up menus is active, the function returns 0. If you provide a pop-up name, dBASE calculates the number of bars, including bars defined with the SKIP option, and returns that value.

## BARCOUNT() BARPROMPT()

### Example

The following example shows how to use the BARCOUNT() function with other functions to obtain and display information on the cursor position in a pop-up menu:

```
*- Set up the display window for the popup information
DEFINE WINDOW BarStat FROM 6,20 TO 10,78 DOUBLE
ACTIVATE WINDOW BarStat
@ 0,2 SAY "Popup status for: "
@ 1,2 SAY " On bar:    of"
@ 2,2 SAY " Prompt:"
ACTIVATE SCREEN

DEFINE POPUP FilePick FROM 3,5 PROMPT FILES LIKE *.*
ON POPUP FilePick DO BarStat      && Display popup info during navigation
ACTIVATE POPUP FilePick
RELEASE WINDOW BarStat
RELEASE POPUP FilePick
RETURN

PROCEDURE BarStat
ACTIVATE WINDOW BarStat
@ 0,20 SAY POPUP()                && Display popup name
@ 1,12 SAY STR(BAR(), 3)          && Display current bar number
@ 1,19 SAY STR(BARCOUNT(), 3)    && Display total number of bars
@ 2,12                                && Clear the previous bar prompt
@ 2,12 SAY LEFT(PROMPT(),35)     && Display the current bar prompt
ACTIVATE SCREEN
RETURN
```

### See Also

ACTIVATE POPUP, BAR(), BARPROMPT(), DEFINE BAR, DEFINE POPUP, ON BAR, ON EXIT BAR, ON EXIT POPUP, ON POPUP, ON SELECTION BAR, ON SELECTION POPUP, PAD(), POPUP(), PROMPT()

---

## BARPROMPT()

BARPROMPT() returns the text that appears in a particular bar of a specified pop-up menu.

### Syntax

BARPROMPT(<expN>[,<expC>])



## Usage

Use BARPROMPT() to find out a bar's prompt text. You must specify the bar number (<expN>), and can optionally specify the prompt name of a specific pop-up menu (<expC>). If you do not specify a pop-up menu, dBASE uses the active pop-up.

BARPROMPT() requires that the pop-up and prompts already be defined in one of the following ways:

1. DEFINE POPUP <popup name> FROM <row>,<column>  
    DEFINE BAR <expN> OF <popup name> PROMPT <expC2>

In the previous example, BARPROMPT(<expN>,<expC>) returns the value of <expC2>.

2. DEFINE POPUP <popup name> FROM <row>,<column>;  
    [PROMPT FIELD <expC>  
    /PROMPT FILES [LIKE <skeleton>]  
    /PROMPT STRUCTURE]

In the previous example, BARPROMPT(<expN>, <expC>) returns the value of the item that appears in the indicated bar. The item could be the value of a field, the name of a file, or the name of a field in the current database file, respectively.

## Example

```
. ? BARPROMPT(10,"Sales")  && Returns text on tenth bar of the Sales popup.  
Report1.frg
```

## See Also

ACTIVATE POPUP, BAR(), BARCOUNT(), DEFINE BAR, DEFINE POPUP, ON BAR, ON EXIT BAR, ON EXIT POPUP, ON POPUP, ON SELECTION BAR, ON SELECTION POPUP, PAD(), PADPROMPT(), POPUP(), PROMPT()

---

## BOF()

The BOF() function indicates the beginning of the file.

### Syntax

BOF([<alias>])

**Usage**

The optional alias parameter identifies a work area and can be a numeric expression from 1 to 40, a character expression from A through J, or a work area name. A work area name can be either a database filename or an alias name specified with the USE command.

BOF() is for applications that read the database in reverse order. It returns a logical true (.T.) when the record pointer is before the first logical record of the file in the work area specified by the alias.

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, USE

## CALL()

CALL() is used together with the LOAD command to execute modules written in other languages (such as FORTRAN or C).

### Syntax

```
CALL(<expC>,<expression>[,<expression list>])
```

### Usage

Like the CALL command, the CALL() function executes the module named by <expC>. Module <expC> is a character expression or memory variable giving the compiled program's filename without its extension. The module must have been loaded with the LOAD command prior to using CALL().

Refer to the LOAD command in Chapter 2 for details on creating and loading modules for the CALL() function.

The expression list may contain up to seven constants, expressions, memory variables, single array elements, or database fields to be passed as parameters to the called module. Memory variables must be defined before they can be passed using the CALL() function. To pass an array element using CALL(), you must declare the array and reference the element's coordinates.

Since a function can return only one value, only the first parameter passed to CALL() is returned. If the remaining values are memory variables, they are updated in memory, but not returned by the CALL() function.

Note that memory variables in <expression> or <expression list> may have been updated in memory by the called routine after the CALL() is done.

### Example

The following example calls a user-written routine, reversec. The memory variable X is created and initialized to blanks. It will receive the result of the REVERSE routine (which will also be the return value of the CALL() function).

```
. LOAD reversec
. X=SPACE(4)
. ? CALL("reversec", X, "ABCD")
DCBA
. ? X
DCBA
```

### See Also

CALL, DECLARE, LOAD, RUN/!, RUN(), STORE

---

## CATALOG()

CATALOG() returns the name of the active catalog file.

### Syntax

CATALOG()

### Usage

dBASE IV does not open the catalog in an accessible work area. Use CATALOG() with the USE command and SELECT() function to place the current catalog in an accessible work area.

If a catalog is not active, CATALOG() returns a null string.

### Examples

If SET FULLPATH is ON, CATALOG() returns the complete path of the catalog file:

```
. SET FULLPATH OFF
. ? CATALOG()
C:CATALOG2.CAT
. SET FULLPATH ON
. ? CATALOG()
C:\MORECATS\CATALOG2.CAT
```

This example opens the current catalog with read-only status in an available work area:

```
. Catname=CATALOG()
C: \MORECATS\CATALOG2.CAT
. USE (Catname) IN SELECT() AGAIN NOUPDATE
```

### See Also

SELECT(), SET CATALOG, SET FULLPATH, SET TITLE

---

## 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(109/9/90)
Sunday
```

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 on a weekend.
```

### 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()

### Default

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, SET LIBRARY, 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 G of this book. Potential returns from CERROR() are marked with an asterisk. CERROR() is not affected by warning messages generated at compile time, such as the following:

**Command not functional in dBASE IV**  
**Extra characters ignored at end of command**  
**Command not valid in RunTime environment**  
**Command will never be reached**

Before executing a new program, use the CERROR() function to test whether the source code compiled successfully and COMPILE command. CERROR() by itself will return zero.

### 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(), CERROR()
  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 LIBRARY, 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

The optional alias parameter identifies a work area and can be a numeric expression from 1 to 40, a character expression from A through J, or a work area name. A work area name can be either a database filename or an alias name specified with the USE command.


## CHANGE() CHR()

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

---

## CHR()

The CHR() function converts a numeric expression to a character.

### Syntax

CHR(<expN>)

### Usage

This function enables you to send any value from the character set to a printer or the screen. <expN> must be an integer from 0 to 255.

Note that not all printers support the entire character set. Check the ASCII code table for your printer if a character from the 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 05/15/91:

```
. ? 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()

CTOD() converts a date that has been entered or stored as a character string to a date type variable.

### Syntax

CTOD(<expC>)

### 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.

## CTOD() DATE()

The format of the character string is normally mm/dd/yy, but this format can be changed by SET CENTURY, SET DATE and SET MARK.

Note that a day value larger than the month increments the month counter and a month value greater than 12 increments the year counter until CTOD() calculates a valid date. See the second example, below.

### 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
```

This example shows how CTOD() converts a character string that would be out of range as a valid date into a date type variable:

```
. ? CTOD("13/32/91")
02/01/92
```

An alternative to using CTOD() is enclosing a literal value in curly braces to create a date variable:

```
. leapdate = {02/29/88}
02/29/88
. SET CENTURY ON
. ? leapdate
02/29/1988
```

### See Also

DATE(), DTOC(), DTOS(), SET CENTURY, SET DATE, SET MARK, TYPE()

---

## 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()  
02/28/91
```

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 05/15/91:

```
. ? 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**

CADOW(), DATE(), DOW()

---

**DBF()**

The DBF() function returns the name of the database file in USE in the specified work area.

**Syntax**

DBF([<alias>])

**Usage**

DBF() is useful in working with a file that is in USE in another work area.

The optional alias parameter identifies a work area and can be a numeric expression from 1 to 40, a character expression from A through J, or a work area name. A work area name can be either a database filename or an alias name specified with the USE command. If you do not specify an alias, DBF() defaults to the current work area.

If no database file is open in the specified work area, DBF() returns a null string.

The return value of DBF() honors the setting of SET FULLPATH.

**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. It returns a logical true (.T.) if the current record in the specified work area is marked for deletion; if it is not, DELETED() returns a logical false (.F.).

### Syntax

DELETED([<alias>])

### Usage

The optional alias parameter identifies a work area and can be a numeric expression from 1 to 40, a character expression from A through J, or a work area name. A work area name can be either a database filename or an alias name specified with the USE command. If you do not specify an alias, DELETED() operates on the current work area.

If no database file is in USE, DELETED() returns a logical false (.F.).

### Example

To determine if the current record is marked for deletion:

```
. USE Client
. ? DELETED()
.F.
. DELETE
1 record deleted.
. ? DELETED()
.T.
```

## DELETED() DESCENDING()

### See Also

PACK, RECALL, SET DELETED

---

## DESCENDING()

This function returns a logical true (.T.) if the specified .mdx index tag was created with the DESCENDING keyword.

### Syntax

```
DESCENDING([[<.mdx filename>.] <expN> [,<alias>]])
```

### Usage

The DESCENDING() function returns a logical true (.T.) if the .mdx tag specified by the optional <expN> parameter was created with the DESCENDING keyword. It returns a logical false (.F.) if the specified tag was created without the DESCENDING keyword.

If the specified .mdx file or tag does not exist, DESCENDING() returns an appropriate error message.

If you do not specify an .mdx filename, DESCENDING interprets <expN> with reference to all open indexes. If the program has a production .mdx file and a non-production .mdx file open, the function checks for the DESCENDING keyword on <expN> in the production .mdx file.

If you do not specify an index tag, DESCENDING() tests whether the controlling tag was created with the DESCENDING keyword.

DESCENDING() returns a logical false (.F.) if the index is an .ndx file.

The optional alias parameter identifies a work area and can be a numeric expression from 1 to 40, a character expression from A through J, or a work area name. A work area name can be either a database filename or an alias name specified with the USE command. If you do not specify an alias, DESCENDING() operates in the current work area.

### Example

```
. USE Client
. INDEX ON Lastname+Initial TAG Nameini DESCENDING
. ? TAGNO("Nameini")
  4
. ? DESCENDING(4)
.T.
```

**See Also**

FOR(), KEY(), MDX(), ORDER(), SET("INDEX"), TAG(), TAGCOUNT(), TAGNO(), UNIQUE()

---

## DGEN()

DGEN() runs the template language interpreter from within dBASE IV programs or at the dot prompt to create dBASE programs from screen, report, label, and menu objects (binary name lists). DGEN() can also pass parameters to the template program that it calls.

**Syntax**

DGEN(<expC1> [,<expC2>])

**Usage**

<expC1> is the name of the template .gen file to use in creating programs. If you omit the filename extension, DGEN() assumes .gen.

The optional <expC2> is an argument string to pass to the template program (.gen file) which you are calling with DGEN().

**Comments**

The Template Language programmer can use the Builtin.def template library functions ARGUMENT() and TOKEN() to parse <expC2>.

DGEN() can return positive and negative values based on the template program's return argument.

DGEN() returns and displays a negative value to indicate an error condition. The negative value corresponds to the positive error message number for Dgen.exe. Refer to *Programming in dBASE IV* for a list of common Dgen errors.

DGEN() returns a -1 in the case of an internal error. DGEN() will display the internal error in its own error box for further diagnosis of the problem.

**Example**

To create a program from Myform.scr in a dBASE program:

```
. DEXPORT SCREEN Myform && Creates Myform.sn1  
. ln_result=DGEN("FORM.GEN","MYFORM.SNL")
```

**See Also**

DEXPORT

See *Programming in dBASE IV*.

## DIFFERENCE()

The DIFFERENCE() function determines the difference between two character expressions.

### 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 = "Kimbreelee"
Kimbreelee
. 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()

DISKSPACE() returns the number of available bytes on the default drive.

### Syntax

DISKSPACE()

### Usage

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.

### Example

```
SET TALK OFF
USE Client
Mfilesize = (RECCOUNT() * RECSIZE()) + 2000
IF DISKSPACE() > Mfilesize
    COPY TO Backup
ELSE
    @ 10,16 SAY "There is not enough room to back up 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**

CROW(), 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 05/13/91:

```
. ? 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

CROW(), DATE(), DAY()

---

## DTC()

The DTC() function converts a date expression to a character string.

### Syntax

DTC(<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 05/13/91):

```
. STORE DTC(DATE()) TO testdate
05/13/91
. ? TYPE("testdate")
C
```

To connect a date variable to a character string for use as text:

```
. STORE DATE() TO testdate
05/13/91
. ? TYPE("testdate")
D
. STORE "The date is: "+DTC(testdate) TO text
The date is: 05/13/91
```

### 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()

EOF() indicates the end of a file. It returns a logical true (.T.) when the last logical record of the specified database file is passed.

### Syntax

EOF([<alias>])

## EOF() ERROR()

### Usage

The optional alias parameter identifies a work area and can be a numeric expression from 1 to 40, a character expression from A through J, or a work area name. A work area name can be either a database filename or an alias name specified with the USE command. If you do not specify an alias, 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.).

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.).

### 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. See Appendix G for a list of error messages, their numbers, and appropriate corrective actions.



**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
RETURN

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. SET DECIMALS affects only the display. The numeric accuracy is not affected.

## EXP() FCLOSE()

### 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

---

## FCLOSE()

FCLOSE() closes a low-level file previously opened with the FCREATE() or FOPEN() functions.

### Syntax

FCLOSE(<expN>)

### Usage

<expN> is a file handle returned by FCREATE() or FOPEN(). FCLOSE() returns a logical true (.T.) if the low-level file is successfully closed. If the close operation is unsuccessful, FCLOSE() returns a logical false (.F.).

dBASE IV blocks any attempt to close a file that was not opened with FCREATE() or FOPEN(). If you specify a file handle not previously returned by FCREATE() or FOPEN(), the **Invalid file handle** error message is displayed.

The CLOSE ALL and CLEAR ALL commands also close any open low-level files.

## Example

```
. Handle=(FCREATE("Fred.txt","RW"))
      3
. ? FPUTS(Handle,"This is a small file.")
      23
. ? FCLOSE(Handle)
.T.
```

## See Also

CLOSE ALL, DISPLAY STATUS, FCREATE(), FEOF(), FERROR(), FFLUSH(), FGETS(), FOPEN(), FPUTS(), FREAD(), FSEEK(), FWRITE()

---

## FCREATE()

FCREATE() creates and opens a low-level file. It returns a file handle number.

### Syntax

FCREATE(<expC1>[,<expC2>])

### Usage

<expC1> specifies the name of the file to create and open. <expC1> can include a full path name for the file and should conform to operating system conventions for the length of filenames and extensions. Filenames that exceed eight characters and extensions that exceed three characters are truncated by DOS.

<expC2> specifies the privilege level and can be one of the following:

- "R" for read-only
- "W" for write only
- "A" for append only
- "RW" or "WR" for read and write
- "AR" or "RA" for read and append

If you do not specify a privilege level, FCREATE() opens the file with read/write privileges.

## FCREATE() FDATE()

FCREATE() returns a positive integer for the file handle. You can create a variable to save this number so that calls to the FCLOSE(), FEOF(), FFLUSH(), FGETS(), FPUTS(), FREAD(), FSEEK(), and FWRITE() functions refer to the same file.

If FCREATE() is unsuccessful, dBASE IV returns an error message and FCREATE() returns no value. Unless dBASE IV intercepts the FCREATE() function call before it reaches the operating system, you can call FERROR() to return the I/O error status.

If you call this function and specify a currently open file that was opened with either FCREATE() or FOPEN(), dBASE IV returns the **File already open** error message.

The DISPLAY STATUS command displays the full path name, file handle number, file size, open mode, and position of the file pointer (relative to the beginning of the file) for all open low-level files.

### Example

```
. Filename="Fred.txt"
. ? FILE(Filename)
.F.
. Handle=FCREATE(Filename,"RA")
. ? FPUTS(Handle,"This is a small file.")
    23
. ? FCLOSE(Handle)
.T.
. ? FOPEN(Filename)
    3
. ? FGETS(Handle)
This is a small file.
```

### See Also

DISPLAY STATUS, FCLOSE(), FEOF(), FERROR(), FFLUSH(), FGETS(), FOPEN(), FPUTS(), FREAD(), FSEEK(), FWRITE()

---

## FDATE()

FDATE() returns the date, from the operating system, that the specified file was last modified.

### Syntax

FDATE(<expC>)

### Usage

<expC> specifies the filename and can include the full pathname. FDATE() searches the current directory and the directories specified with the SET PATH command.

FDATE() does not support wildcard characters in <expC>.

FDATE() returns the date in the form mm/dd/yy. This format can be modified with the SET CENTURY, SET DATE, and SET MARK commands.

To ensure the most accuracy, close the file specified by <expC> before calling FDATE().

### Example

```
. ? FDATE("Employee.dbf")
05/18/91
. SET DATE TO British
. ? FDATE("Employee.dbf")
18/05/91
```

### See Also

DISKSPACE(), FILE(), FSIZE(), FTIME(), SET CENTURY, SET DATE, SET MARK

## FEOF()

FEOF() tests whether the file pointer is at the end of a low-level file.

### Syntax

FEOF(<expN>)

### Usage

<expN> is a file handle returned by FCREATE() or FOPEN() for a currently open file. FEOF() returns a logical true (.T.) if the pointer is at the end of the low-level file specified by <expN>. If the file pointer is not at the end of the file, FEOF() returns a logical false (.F.).

If you specify a file handle not previously returned by FCREATE() or FOPEN(), the **Invalid file handle** error message is displayed.

### Example

This example uses FEOF() to test for the end of a file whose contents are being read and displayed one line at a time:

```
*- Read each line of the file and display it
lh_txt=0
lh_txt = FOPEN( "Fred.txt", "R" )      && Open Fred.txt for reading
IF lh_txt > 0                          && If the file was opened OK
    DO WHILE .NOT. FEOF( lh_txt )      && Loop while FEOF() is not true
        ? FGETS( lh_txt )             && Read the line and output it
    ENDDO
ENDIF
```

### See Also

FCLOSE(), FCREATE(), FERROR(), FFLUSH(), FGETS(), FOPEN(), FPUTS(),  
FREAD(), FSEEK(), FWRITE()

---

## FERROR()

FERROR() returns the operating system error status number for a low-level file operation.

### Syntax

FERROR()

### Usage

FERROR returns a positive integer for the operating system error code when a preceding low-level file I/O function fails at the operating system level.

Typical DOS I/O error codes include:

- 2 File not found
- 3 Path not found
- 4 Too many open files (no handles left)
- 5 Access denied
- 6 Invalid handle
- 8 Insufficient memory
- 25 Seek error
- 29 Write fault
- 31 General failure

FERROR() returns a zero to indicate that a previously called low-level file function completed successfully or never executed. FERROR() can also return a zero when the low-level file function was intercepted by dBASE IV and did not reach the operating system. To determine if this is the case when a zero is returned from FERROR(), use the ERROR() function.



### Examples

This example is a subroutine that uses FERROR() to determine which error message should be displayed:

```

PROCEDURE DOSIOERR
  Errno=FERROR()
  DO CASE
    CASE Errno=0
      ? "There was no error"
    CASE Errno=2
      ? "File doesn't exist"
    CASE Errno=3
      ? "Path not found"
    CASE Errno=4
      ? "No more handles available"
    CASE Errno=5
      ? "Access denied"
    CASE Errno=8
      ? "Buffer full"
    CASE Errno=31
      ? "Error opening file - general failure"
  END CASE
RETURN

```

### See Also

ERROR(), FCLOSE(), FCREATE(), FEOF(), FFLUSH(), FGETS(), FOPEN(), FPUTS(), FREAD(), FSEEK(), FWRITE()

## FFLUSH()

FFLUSH() writes the contents of the system buffer for a specified low-level file to disk.

### Syntax

FFLUSH(<expN>)

### Usage

<expN> is a file handle returned by FCREATE() or FOPEN(). FFLUSH() returns a logical true (.T.) if the low-level file is successfully written to disk. If the flush operation is unsuccessful, FFLUSH() returns a logical false (.F.).

The low-level file is written to disk under the name specified in the FCREATE() or FOPEN() function call that returned the file handle used with FFLUSH().

To be certain that FFLUSH() updates the disk file, disable any delayed writing features of disk cache programs.

**Example**

```
. Handle=(FCREATE("Fred.txt","A"))
      4
. ? FPUTS(Handle,"This is a small file.")
      23
. ? FFLUSH(Handle)
.T.
. ? FPUTS(Handle,"With two lines of text.")
      25
. ? FCLOSE(Handle)
.T.
```

**See Also**

FCLOSE(), FCREATE(), FEOF(), FERROR(), FGETS(), FOPEN(), FPUTS(), FREAD(), FSEEK(), FWRITE()

---

**FGETS()**

FGETS() reads and returns a string of characters from a low-level file opened with the FCREATE() or FOPEN() functions.

**Syntax**

FGETS(<expN1>[,<expN2>][,<expC>])

**Usage**

<expN1> is a file handle returned by FCREATE() or FOPEN(). If you specify a file handle not previously returned by FCREATE() or FOPEN(), the **Invalid file handle** error message is displayed.

The optional <expN2> parameter specifies the number of bytes to be read by FGETS() from the current position of the file pointer. <expN2> can be between 0 and 254. If you do not specify the number of bytes to be read, FGETS() defaults to 254.

Use the optional <expC> parameter to define the end of line indicator. <expC> must evaluate to one or two characters. If you do not specify an end-of-line indicator, FGETS() reads the specified number of characters or until it encounters a Carriage Return and a Line Feed, 0D0A Hex, specified as *CHR(13)+CHR(10)*.

Other possible values for the end-of-line indicator are a Soft Carriage Return, 8D Hex (US) or FF Hex (Europe) and a Soft Line Feed, 8A Hex (US), 00 (Europe). Note that dBASE IV does not discard a single CHR(0) when passed to or from the low-level file functions. You can pass one CHR(0) at a time.

The character string returned by FGETS() does not include end-of-line indicators. FGETS() places the file pointer past the end-of-line indicator or, if an end-of-line indicator is not encountered, past the last character read. If you want Soft Carriage Return or Soft Line Feed characters included in the return string, specify <expC> to define only soft end-of-line indicators or use the FREAD() function.

Note that FGETS() truncates the return string at the first 00 Hex character, CHR(0), it encounters. If you suspect that the file contains 00 Hex characters, use FREAD() and read one byte at a time.

FGETS() returns a null string if it is unsuccessful (as when the file pointer is at the end of the file). You can use the FEOF() and FERROR() functions for error checking in such situations.

If the file specified by <expN1> was not opened with read privileges, dBASE IV returns the **Read error** error message.

### Examples

```
. Handle=FOPEN("Fred.txt","R")
  4
. ? FGETS(Handle)
This is a small file.
. ? FCLOSE(Handle)
.T.
```

```
Ghandle=FOPEN("Book.txt")
IF (Ghandle > 0)
  DO WHILE .NOT. FEOF (Ghandle)
    Line=FGETS(Ghandle)    && Read a line from Book.txt
    ? Line                 &&Display the line
  ENDDO
ENDIF
Null=FCLOSE(Ghandle)
```

### See Also

FCLOSE(), FCREATE(), FEOF(), FERROR(), FFLUSH(), FOPEN(), FPUTS(), FREAD(), FSEEK(), FWRITE()

## FIELD()

FIELD() returns the field name of the specified field number from the file structure of the selected database file.

## FIELD() FILE()

### Syntax

FIELD(<expN>[,<alias>])

### Usage

The optional alias parameter identifies a work area and can be a numeric expression from 1 to 40, a character expression from A through J, or a work area name. A work area name can be either a database filename or an alias name specified with the USE command. If you do not specify an alias, FIELD() operates on the current work area.

FIELD() 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, dBASE IV returns a null string.

Because the FIELD() function addresses each field by its number, it enables you to handle fields as an array.

FIELD() is not affected by the SET FIELDS command.

### Examples

Use the Client database file.

```
. USE Client
. ? FIELD(3)
  LASTNAME
. number = 2
      2
. Mfield = FIELD(number)
  CLIENT
```

### See Also

DBF(), FLDCOUNT(), LUPDATE(), RECCOUNT(), RECSIZE(), TYPE(), VARREAD()

---

## FILE()

FILE() tests for the existence of a specified filename. It returns a logical true (.T.) if the file exists.

### Syntax

FILE(<expC>)

**Usage**

The FILE() function requires both the filename and its extension. If the file is not in the current directory or the directories specified with the SET PATH command, you must give the complete path.

The FILE() function is not case sensitive.

**Examples**

If the file client.dbf is not in the current directory, you must provide the full path or use SET PATH:

```
. ? FILE("Client.dbf")
.F.
. SET PATH D:\Dbase\Samples
. ? FILE("Client.dbf")
.T.
```

FILE() does not find the file without the extension:

```
. ? FILE("D:\Dbase\Samples\Client")
.F.
```

Specify the complete filename, with extension, and the path:

```
. ? FILE("D:\Dbase\Samples\Client.dbf")
.T.
```

**See Also**

SET DEFAULT, SET DIRECTORY, SET PATH

**FIXED()**

The FIXED() function converts floating point numbers (Type F) to Binary Coded Decimal numbers (Type N).

**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.

**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 with dBASE IV.*)

The FKLABEL() function returns the key label associated with the function key specified by <expN>. Any numeric expression from 1 to 28 is valid.

The **F1 Help** 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, ON KEY

---

## FKMAX()

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 under DOS.

dBASE IV uses the **F1 Help** key for 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, ON KEY

---

## FLDCOUNT()

FLDCOUNT() returns the number of fields in the structure of the specified database.

**Syntax**

FLDCOUNT([&lt;alias&gt;])

**Usage**

The optional alias parameter identifies a work area and can be a numeric expression from 1 to 40, a character expression from A through J, or a work area name. A work area name can be either a database filename or an alias name specified with the USE command. If you do not specify an alias, FLDCOUNT() returns the number of fields in the database file in USE in the current work area.

If no database is in use, FLDCOUNT() returns zero.

**Example**

```
. USE Client  
. ? FLDCOUNT()  
10
```

**See Also**

FIELD(), LIST/DISPLAY STRUCTURE, RECCOUNT(), RECSIZE()

---

## **FLDLIST()**

FLDLIST() returns the fields and calculated field expressions of a SET FIELDS TO list.

**Syntax**

FLDLIST([<expN>])

**Usage**

Use FLDLIST() to scan the contents of a SET FIELDS TO list. For example, to find out the name of the first field or expression in the SET FIELDS TO list, use FLDLIST(1).

If you omit <expN>, FLDLIST() returns the entire field list, up to 254 characters. All characters beyond that limit are truncated. Each field name or expression is separated by a comma.

FLDLIST() always returns a fully-qualified field name, i.e., it includes the file or alias name. It returns the calculated expression for a calculated field, and appends “/R” to a read-only field.

<expN> must be an integer greater than 0. If <expN> exceeds the number of items in the fields list, or if you cleared the list (using SET FIELDS TO), FLDLIST() returns a null string.

FLDLIST() returns a value even if SET FIELDS is OFF.



### Example

```

. USE Myfile IN 1                && Assume Myfile contains these fields:
                                && Name, Address, Age
. USE Report ALIAS Rep IN 2     && Assume Report contains these fields:
                                && Date, Status

. SELECT 1
. SET FIELDS TO Name, Age/R     && Add items to the SET FIELDS TO list
. ? FDLIST(1)                   && Return first item in SET FIELDS TO list
MYFILE->NAME
. SELECT 2
. SET FIELDS TO Date           && Add another item to the SET FIELDS TO
list
. ? FDLIST(1)
MYFILE->NAME
. ? FDLIST(2)
MYFILE->AGE/R
. ? FDLIST(3)
REP->DATE                       && Note that the alias is used
. SET FIELDS TO               && Clear the list
. SELECT 1
. SET FIELDS TO Newage=Age*2   && Add calculated field to the list.
. ? FDLIST(1)
NEWAGE = Age*2

```

### See also

SET FIELDS

---

## FLOAT()

The `FLOAT()` function converts Binary Coded 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 range allowed is between  $0.9 \times 10^{-307}$  and  $0.9 \times 10^{-308}$ . The range for type F numbers is the same.

Any expression containing a type F number returns a type F result.

**See Also**

FIXED(), SET PRECISION

---

## **FLOCK()**

FLOCK() 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 specific records 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.

The optional alias parameter identifies a work area and can be a numeric expression from 1 to 40, a character expression from A through J, or a work area name. A work area name can be either a database filename or an alias name specified with the USE command. If you specify an alias, FLOCK() attempts to lock the file in the designated work area. If you do not specify an alias, FLOCK() locks the file in the current work area.

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 in a multi-user environment.

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
. ? FLOOR(First / -10)
      -13
```

**See Also**

CEILING(), INT(), MOD(), ROUND()

## FOPEN()

FOPEN() opens an existing file and returns a file handle number.

### Syntax

FOPEN(<expC1>[,<expC2>])

### Usage

<expC1> specifies the name of the file to open. <expC1> can include a full path name for the file. FOPEN() searches the current directory and the directories specified with the SET PATH command.

<expC2> specifies the privilege level. The initial position of the pointer within the opened file depends upon the privilege level. Privilege levels and the relative positions of the file pointer are:

Privilege level	File pointer position
"R" for read only	Beginning of file
"W" for write only	Beginning of file
"A" for append only	End of file
"RW" or "WR" for read and write	Beginning of file
"AR" or "RA" for read and append	End of file

If you do not specify a privilege level, FOPEN() opens the file with read only privileges.

FOPEN() returns a positive integer for the file handle that can be used in calls to other low-level file functions. If FOPEN() is unsuccessful, it returns no value. Use the ERROR() function to examine an error returned by dBASE IV or the FERROR() function to examine an error returned by the operating system.

Use the FSEEK() function to position the file pointer before writing to a file opened with write privileges.

The DISPLAY STATUS command displays the full path name, file handle number, file size, open mode, and position of the file pointer (relative to the beginning of the file) for all open low-level files.

If you call this function and specify a currently open file that was opened with either FCREATE() or FOPEN(), dBASE IV returns the **File already open** error message in networked environments (if NETWORK() = .T.).

Use the FCLOSE() function to close a file opened with FCREATE() or FOPEN(). The CLEAR ALL and CLOSE ALL commands close all files opened with FOPEN() or FCREATE().

### Example

```

Handle=0                                && Predefine handle variable
IF FILE("Myfile.txt")                   && Check if file exists
    Handle=FOPEN("Myfile.txt", "R") && Open for read only
ENDIF
IF Handle > 0                            && Did file open OK?
    DO WHILE .NOT. FEOF(Handle)
        String=FGETS(Handle)
        ? String
    ENDDO
ELSE
    ? "Unable to open file. File error #:",FERROR()
ENDIF
IF FCLOSE(Handle)
    ?
    ? "File was closed."
ELSE
    ? "Unable to close file. File error #:",FERROR()
ENDIF

```

### See Also

CLOSE ALL, DISPLAY STATUS, ERROR, FCLOSE(), FCREATE(), FEOF(), FERROR(), FFLUSH(), FGETS(), FPUTS(), FREAD(), FSEEK(), FWRITE()

---

## FOR()

FOR() returns the FOR condition used to create an .mdx file index tag.

### Syntax

```
FOR([[<.mdx filename>], <expN> [,<alias>]])
```

### Usage

FOR() returns the FOR clause used to create the .mdx tag specified by the optional <expN> parameter. If the optional .mdx filename is specified, <expN> refers to that file.

If you do not specify an index tag, FOR() returns the FOR condition of the controlling index.

If no .mdx filename is specified, FOR() interprets <expN> with reference to all open indexes in the work area and checks .mdx file expressions first (if an .mdx file exists, FOR() returns a null string). FOR() next checks production .mdx tags and then non-production .mdx tags.

## FOR()

The optional alias parameter identifies a work area and can be a numeric expression from 1 to 40, a character expression from A through J, or a work area name. A work area name can be either a database filename or an alias name specified with the USE command. If you do not specify an alias, FOR() operates in the current work area.

FOR() returns a null string when:

- applied to an .ndx file expression
- the tag does not exist
- the tag does not contain a FOR condition or clause

Because FOR() returns the FOR clause as it was entered by the user, your program should perform any required string comparisons on the return value.

### Example

This example assumes DISPLAY STATUS shows the following indexes in use:

```
. DISPLAY STATUS
Currently Selected Database:
Select area: 1, Database in Use: D:\STAFF\EMPLOYEE.DBF Alias: MYDATA
      Index file: D:\STAFF\LASTNAME.NDX Key: lastname
      Index file: D:\STAFF\CITY.NDX Key: city
Production MDX file: D:\STAFF\EMPLOYEE.MDX
      Index TAG: TEMP1 Key: lastname For: lastname="Smith"
      Index TAG: DEPT Key: department+lastname+firstname+initial
      Index TAG: TITLESPEC Key: TRIM(TITLE)+SPECIALTY
      Index TAG: FIRSTNAME Key: FIRSTNAME
Master Index TAG: CITY Key: CITY
      Index TAG: DEPTTITLE Key: UPPER(Department+Title)
      Index TAG: NAMES Key: lastname+firstname+initial
      Index TAG: STATE Key: STATE
      Index TAG: ZIP Key: ZIP
      Index TAG: LASTNAME Key: LASTNAME
      Index TAG: EMP_ID Key: emp_id For: LIKE("1*",emp_id)
      MDX file: D:\STAFF\FRED\MORETAGS.MDX
      Index TAG: NEWTAG Key: lastname
      Index TAG: CALIF Key: State For: State="CA"

. ? FOR(3)
lastname="Smith"
. ? FOR("Employee",1)
lastname="Smith"
. ? FOR("Moretags",2)
State="CA"
```

### See Also

DESCENDING(), INDEX, KEY(), SET("INDEX"), TAG(), TAGCOUNT(), TAGNO(), UNIQUE()

## FOUND()

FOUND() returns a logical true (.T.) if the previous FIND, LOCATE, SEEK, or CONTINUE command or LOOKUP() or SEEK() function 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. The optional alias parameter identifies a work area and can be a numeric expression from 1 to 40, a character expression from A through J, or a work area name. A work area name can be either a database filename or an alias name specified with the USE command. If you do not specify an alias, 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 the FIND, LOCATE, SEEK, or CONTINUE commands or the SEEK() or LOOKUP() functions.

If you move the record pointer with any command other than FIND, LOCATE, SEEK, or CONTINUE or function other than SEEK() or LOOKUP(), the result of FOUND() is a logical false (.F.).

When SET NEAR is ON:

- FOUND() returns a true (.T.) if an exact match has occurred
- FOUND() returns a false (.F.) for a near match, and the pointer is moved to the record whose key sorts immediately next to the sought value

When SET NEAR is OFF, FOUND() returns a false (.F.) if a match does not occur.

### 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()7
.F.
```

**See Also**

CONTINUE, EOF(), FIND, LOCATE, LOOKUP(), SEEK, SEEK(), SET NEAR, SET RELATION

---

## FPUTS()

FPUTS() writes a specified character string, including an end-of-line indicator, to a specified file at the current position of the file pointer. It returns the number of bytes written to the file.

**Syntax**

FPUTS(<expN1>,<expC1>[,<expN2>][,<expC2>])

**Usage**

<expN1> is a file handle returned by FCREATE() or FOPEN(). If you specify a file handle not previously returned by FCREATE() or FOPEN(), dBASE IV returns the **Invalid file handle** error message.

<expC1> specifies the character string to be written by FPUTS(), beginning at the current location of the file pointer. FPUTS() moves the file pointer past the last character written to the file.

The optional <expN2> parameter specifies the number of bytes from the character string to be written by FPUTS(). <expN2> can be between 0 and 254. If <expN2> is less than 0, FPUTS() uses 0; if greater than 254, FPUTS() uses 254.

If you do not want FPUTS() to write the default end-of-line indicator to the file, use the optional <expC2> parameter to define an end-of-line indicator. <expC2> must evaluate to one or two characters. The default end-of-line indicator used by FPUTS() is a Carriage Return/Line Feed (0D0A Hex). See the FGETS() function for more information about end-of-line indicators.

Use the FWRITE() function if you do not want an end-of-line indicator written to the file.

FPUTS() returns the number of bytes written to the file. This number includes the end-of-line indicator. If FPUTS() is unsuccessful, it returns a zero. Use FERROR() to determine which type of error occurred.



## Example

```

Cityfile="D:\City\City.txt"
Handle=0                && Predefine handle variable
IF .NOT. FILE(Cityfile)  && Check if file exists
    Handle=FCREATE(Cityfile,"A") && Create and open file to append data
ENDIF
IF Handle < 1           && If the positive handle exists
    ?? "File cannot be created. Error #", FERROR()
    RETURN
ENDIF
USE D:\Dbase\Samples\Client
NumBytes=0
SCAN
    NumBytes=FPUTS(Handle,RTRIM(City))+NumBytes
ENDSCAN
? NumBytes, " characters were written to file: ", Cityfile
NULL=FCLOSE(Handle)
RETURN

```

## See Also

FCLOSE(), FCREATE(), FEOF(), FERROR(), FFLUSH(), FGETS(), FOPEN(), FREAD(), FSEEK(), FWRITE()

---

## FREAD()

FREAD() reads and returns a string of characters, including any end-of-line indicators, from a file opened with the FCREATE() or FOPEN() functions.

### Syntax


FREAD(<expN1>,<expN2>)

### Usage

<expN1> is a file handle returned by FCREATE() or FOPEN(). If you specify a file handle not previously returned by FCREATE() or FOPEN(), dBASE IV returns the **Invalid file handle** error message.

<expN2> specifies the number of bytes to be read by FOPEN() from the current position of the file pointer. <expN2> can be between 0 and 254, inclusive. If <expN2> is less than 0, FREAD() uses 0; if greater than 254, FREAD() uses 254. Use the FSEEK() function to position the file pointer, if required, before calling FREAD().

The character string returned by FREAD() can include end-of-line indicators. FREAD() places the file pointer past the last character read.

 **NOTE** *Not all systems use the same high-low convention for two-byte data. Keep in mind that the high-low byte order may differ on a platform to which you port your program.*

FREAD() reads the specified number of bytes or until it encounters an end-of-file marker. FREAD() does not include an end-of-file marker in its return string.

FREAD() truncates the return string at the first 00 Hex character, CHR(0), it encounters. If you suspect that the file contains 00 Hex characters, read one byte at a time.

If FREAD() is unsuccessful (as when the file pointer is at the end of the file), it returns a null string. You can use the FEOF() and FERROR() functions for error checking in such situations.

## Examples

```
. Handle=FOPEN("Contents.dbt","R")
. ? FSEEK(Handle,520)
    520
. ? FREAD(Handle,54)
Remember to send Ralph and Alice a
thank-you note.
```

```
Handle=0
IF FILE("Foo.txt")
    Handle=FOPEN("Foo.txt")
ENDIF
IF Handle < 1
    ? "Cannot open file: Foo.txt. Error #",FERROR()
    RETURN
ENDIF
USE A.DBF
ZAP
DO WHILE .NOT. FEOF(Handle)
    Manuf=FREAD(Handle,30)
    IF LEN(Manuf)=30
        APPEND BLANK
        REPLACE Company WITH Manuf
    ENDIF
ENDDO
NULL=FCLOSE(Handle)
RETURN
```

## See Also

ERROR(), FCLOSE(), FCREATE(), FEOF(), FERROR(), FFLUSH(), FGETS(), FOPEN(), FPUTS(), FSEEK(), FWRITE()

Refer to *Programming in dBASE IV* for examples of user-defined functions that use FREAD().

## FSEEK()

FSEEK() moves the file pointer to a specified location within a file opened by FCREATE() or FOPEN().

### Syntax

```
FSEEK(<expN1>,<expN2>[,<expN3>])
```

### Usage

<expN1> is a file handle returned by FCREATE() or FOPEN(). If you specify a file handle not previously returned by FCREATE() or FOPEN(), dBASE IV returns the **Invalid file handle** error message.

<expN2> specifies the number of bytes to move the file pointer. This number can be between  $-(2^{31})+1$  to  $(2^{31})-1$  (between +2 billion and -2 billion bytes). A positive number moves the file pointer toward the end of the file and a negative number moves the pointer toward the beginning of the file.

The optional <expN3> parameter specifies the starting location of the file pointer. Valid starting position values are: 0 for the beginning of the file, 1 for the pointer's current location, and 2 for the end of the file. If you do not specify a starting location with <expN3>, FSEEK() uses a default value of 0 and <expN2> is relative to the beginning of the file.

FSEEK() returns the final location of the file pointer, relative to the beginning of the file.

FGETS() and FREAD() place the file pointer past the last character read. FPUTS() and FWRITE() place the file pointer past the last character written. The DISPLAY STATUS command displays the position of the file pointer (relative to the beginning of the file) for all open low-level files.

### Examples

```
NewLoc=FSEEK(Handle,0,2)      && Move to EOF
NewLoc=FSEEK(Handle,0,0)      && Move to BOF (rewind)
NewLoc=FSEEK(Handle,0)        && Move to BOF
NewLoc=FSEEK(Handle,512,0)    && Move to byte 512
NewLoc=FSEEK(Handle,16,1)     && Move 16 bytes forward
                             && from the current location
NewLoc=FSEEK(Handle,-16,1)    && Move sixteen bytes backward
                             && from current location
NewLoc=FSEEK(Handle,0,1)     && Return current location
```

### See Also

DISPLAY STATUS, FCLOSE(), FCREATE(), FEOF(), FERROR(), FFLUSH(), FGETS(), FOPEN(), FPUTS(), FREAD(), FWRITE()

---

## **FSIZE()**

FSIZE() returns an integer for the size, in bytes, of the specified file as reported by the operating system.

### **Syntax**

FSIZE(<expC>)

### **Usage**

<expC> specifies the filename and can include the full pathname. FSIZE() searches the current directory and the directories specified with the SET PATH command.

FSIZE() does not support wildcard characters in <expC>.

The file specified by <expC> may have been modified during the dBASE IV session. To ensure accuracy, close or flush the file before calling FSIZE().

### **Example**

```
. ? FSIZE("Employee.dbf")  
13022
```

### **See Also**

DISKSPACE(), FDATE(), FILE(), FTIME()

---

## **FTIME()**

FTIME() returns the time, from the operating system, that the specified file was last modified.

### **Syntax**

FTIME(<expC>)

### **Usage**

<expC> specifies the filename and can include the full pathname. FTIME() searches the current directory and the directories specified with the SET PATH command.

FTIME() does not support wildcard characters in <expC>.

FTIME() returns the time in a character string in the form hh:mm:ss. The SET HOURS TO setting does not alter the format of the string returned by FTIME().

The file specified by <expC> may have been modified during the dBASE IV session. To ensure accuracy, close or flush the file before calling FTIME().

### Example

```
. ? FTIME("Employee.dbf")
10:11:07
```

### See Also

DISKSPACE(), FILE(), FDATE(), FSIZE()

## FV()

FV() 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 the effective interest rate per period, expressed as a positive decimal number. If you are calculating a monthly payment using an annual interest rate, divide the rate by 12. For example, .015 represents an 18% annual interest rate.

<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(), PAYMENT()

---

## FWRITE()

FWRITE() writes a specified character string to a file opened with FCREATE() or FOPEN(). It returns the number of bytes written to the file.

### Syntax

```
FWRITE(<expN1>, <expC1>[, <expN2>])
```

### Usage

<expN1> is a file handle returned by FCREATE() or FOPEN(). If you specify a file handle not previously returned by FCREATE() or FOPEN(), dBASE IV returns the **Invalid file handle** error message.

<expC1> specifies the character string to be written by FWRITE(), beginning at the current location of the file pointer. FWRITE() moves the file pointer past the last character written to the file. Note that FWRITE() writes only one 00 Hex character, CHR(0), at a time. You need to send the 00Hex characters one at a time to write as many zeros as you require to your low-level file.

The optional <expN2> parameter specifies the number of bytes from the character string to be written to the file. <expN2> can be between 0 and 254, inclusive. If <expN2> is less than 0, FWRITE() uses 0; if greater than 254, FWRITE() uses 254. If you do not specify the number of bytes to be written to the file, FWRITE() writes the entire character string. You can use a value of 0 for <expN2> to truncate a file from the position of the file pointer. If you call FWRITE() with <expN2> set to 0 and then close the file, all data beyond the position of the file pointer is discarded.

Use the FSEEK() function to position the file pointer, if required, before calling FWRITE().

FWRITE() returns the number of bytes written to the file. If FWRITE() is unsuccessful, it returns a zero. Use the FERROR() and ERROR() functions to determine which type of error occurred.

**Example**

```
Nfile="Names.txt"
Handle=FCREATE(Nfile,"RW")  && Create and open file
IF Handle < 0
    ?? "... File cannot be created. Error #", FERROR()
    RETURN
ENDIF
USE D:\Dbase\Samples\Client
NumBytes=0
SCAN
    NumBytes=FWRITE(Handle,RTRIM(Lastname)+", "+RTRIM(Firstname);
    +CHR(13)+CHR(10))+NumBytes
ENDSCAN
? NumBytes, "characters written to", Nfile
NULL=FCLOSE(Handle)
RETURN
```

**See Also**

**ERROR(), FCLOSE(), FCREATE(), FEOF(), FERROR(), FFLUSH(), FGETS(), FOPEN(), FPUTS(), FREAD(), FSEEK()**

Refer to *Programming in dBASE IV* for examples of user-defined functions that use FWRITE().

**GETENV()**

GETENV() returns the contents of a system environment variable.

**Syntax**

GETENV(<expC>)

**Usage**

GETENV() returns the setting of a specified environment variable, such as PATH, PROMPT, or COMSPEC. If GETENV() does not find the specified environment variable, it returns a null string.

The name of the variable may be entered as a character string, a character variable, or a combination of both.

## GETENV() HOME()

### Example

To find the current path setting, check the environmental variable PATH:

```
. ? GETENV("PATH")  
C:\;C:\DOS;C:\BIN;C:\DBASE
```

### See Also

OS()

---

## HOME()

HOME() returns the path from which the current dBASE IV session was invoked.

### Syntax

HOME()

### Usage

HOME() returns a character string for the drive and directory path to the dBASE IV system files that were invoked for the current session.

### Example

```
. ? HOME()  
D:\DBASE\
```

### See Also

ID(), NETWORK(), OS()



---

## ID()

ID() returns the name of the current user on a Local Area Network or other multi-user system.

### Syntax

ID()

### Usage

ID() takes no arguments and returns the name of the current user in a character string.

ID() returns a null string when called on a single-user system or a user name is not registered.

### Example

```
Usr=ID()
IF LEN(Usr)=0
    ? "... No user is logged on."
    RETURN
ENDIF
? Usr, " is currently logged on."
RETURN
```

### See Also

LIST USERS, NETWORK(), USER()

---

## IIF()

IIF() stands for *immediate IF* and is a shortcut to the IF...ENDIF programming construct.

### Syntax

IIF(<condition>,<exp1>,<exp2>)

### Usage

If the logical expression specified by <condition> 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. They do not have to be the same data type.

The IIF() function allows you to replace a field or a value in an expression with one of two values, based on the evaluation of a condition.

## Examples

In the following program, you can replace the first set of commands with the one IIF() command line that immediately follows them. This example uses the IF...ENDIF structure:

```
IF Sex = "F"
  Mname = "Ms. " + Last_name
ELSE
  Mname = "Mr. " + Last_name
ENDIF
```

This is the one-line replacement using IIF():

```
Mname = IIF(Sex = "F", "Ms. ", "Mr. ") + Last_name
```

From within the CREATE LABEL menu, you could use this IIF() construction to design and print labels with the appropriate title by including IIF() in a calculated field.

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()

INKEY() returns an integer representing the most recent key pressed by the operator. If the numeric argument is not specified, INKEY() does not halt program execution.

### Syntax

INKEY([<expN>])

### Usage

INKEY() is used as a condition for branching. It lets you use function and cursor movement keys within a dBASE IV program.

INKEY() returns:

- an integer between 0 and 255, corresponding to the decimal ASCII value of the character in the type-ahead buffer (refer to the ASCII chart in Appendix E)
- the control-key equivalent of special keys, such as arrow keys and **Ctrl-Home**
- 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. INKEY() also returns a zero in response to certain key combinations, such as **Ctrl-0** and **Ctrl-9**. You should determine the value a particular keystroke will return before including INKEY() in a program segment.

Returned decimal codes for all keys and key combinations (except the **Alt** key combinations) are listed in Table 4-1. INKEY() (without a parameter) 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().

Table 4-1 INKEY() returned values

Key Names		Decimal value
Ctrl-0		-404
Ctrl-1		-404
Ctrl-2		-404
Ctrl-3		-404
Ctrl-5		-404
Ctrl-6		30
Ctrl-7 through Ctrl-9		-404
Ctrl--		-403
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

*(continued)*

Table 4-1 INKEY() returned values (continued)

Key Names		Decimal value
Ctrl-V	Ins	22
Ctrl-W	Ctrl-End	23
Ctrl-X	↓	24
Ctrl-Y		25
Ctrl-Z	Home	26
Esc	Ctrl-[	27
F1	Ctrl-\	28
Ctrl-]	Ctrl-Home	29
Ctrl-^, Ctrl-6	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

(continued)

Table 4-1 INKEY() returned values (continued)

Key Names	Decimal value
<b>Ctrl-F6</b>	-15
<b>Ctrl-F7</b>	-16
<b>Ctrl-F8</b>	-17
<b>Ctrl-F9</b>	-18
<b>Ctrl-F10</b>	-19
<b>Shift-F1</b>	-20
<b>Shift-F2</b>	-21
<b>Shift-F3</b>	-22
<b>Shift-F4</b>	-23
<b>Shift-F5</b>	-24
<b>Shift-F6</b>	-25
<b>Shift-F7</b>	-26
<b>Shift-F8</b>	-27
<b>Shift-F9</b>	-28

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 4-2 **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 ESCAPE, SET TYPEAHEAD

**INT()**

The INT() function truncates any numeric expression to an integer.

**Syntax**

INT(<expN>)

**Usage**

You can discard all digits to the right of the decimal point in a numeric expression by using INT().

**Example**

To convert the number 10.23 to an integer:

```
. ? INT(10.23)
      10
. STORE 10.23 TO x
      10.23
. STORE INT(10.23) TO x
      10
```

**See Also**

CEILING(), FLOOR(), ROUND()

---

## ISALPHA()

The ISALPHA() function returns a logical true (.T.) if the specified character expression begins with an alphabetical character.

**Syntax**

ISALPHA(<expC>)

**Usage**

<expC> is any character string.

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("123abc")  
.F.
```

**See Also**

ISLOWER(), ISUPPER(), LOWER(), UPPER()

---

## ISBLANK()

ISBLANK() tests whether the specified expression is blank.

**Syntax**

ISBLANK(<exp>)

**Usage**

ISBLANK() returns a logical true (.T.) if the valid expression specified by <exp> evaluates to a blank value of any type. If <exp> is not blank, ISBLANK() returns a logical false (.F.).



Use ISBLANK() to check if a field, memory variable, array element, or other expression evaluates to blank. Blank values for each data type are listed in Table 2-5 under the BLANK command. A blank memo field has no pointer in the .dbf file and shows the lowercase "memo" marker.

### Examples

```
. ? ISBLANK(" ")
.T.
. ? ISBLANK(" / / ")
.F.
. ? ISBLANK({ / / })
.T.
. X = ""
. ? ISBLANK(X)
.T.
```

When applied to a field of a new record created with APPEND BLANK, ISBLANK() returns true (.T.):

```
. DISPLAY
Record #   SNO   SFIRSTNAME   SLASTNAME   Commission
         8     9   Joe         Schectman   50000
. APPEND BLANK
. ? ISBLANK(Commission)
.T.
```

Use ISBLANK() to control how calculations are performed with blank data:

```
. CALCULATE AVG(Commission)
      9 records
AVG(Commission)
      36000

. CALCULATE AVG(Commission) FOR .NOT. ISBLANK(Commission)
      8 records
AVG(Commission)
      40500
```

### See Also

APPEND [BLANK], BLANK, ISALPHA(), SPACE()

---

## **ISCOLOR()**

ISCOLOR() returns a logical true (.T.) if a computer is capable of displaying color.

### **Syntax**

ISCOLOR()

### **Usage**

ISCOLOR() enables developers to design an application for use in both color and 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 monochrome monitor is run by a color card, as in COMPAQ portables. The display may be in shades of grey instead of color.

### **Example**

Execute the following commands in a dBASE IV program to set the color environment:

```
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

*Getting Started with dBASE IV* explains how to change settings in Config.db.

---

## 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.

## ISMARKED() ISMOUSE()

The optional alias parameter identifies a work area and can be a numeric expression from 1 to 40, a character expression from A through J, or a work area name. A work area name can be either a database filename or an alias name specified with the USE command.

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.

### See Also

BEGIN/END TRANSACTION, COMPLETED(), RESET, ROLLBACK, ROLLBACK()

---

## ISMOUSE()

ISMOUSE() returns a logical true (.T.) or false (.F.) to indicate whether or not a mouse driver has been installed.

### Syntax

ISMOUSE()

### Usage

Use this function to determine if a mouse driver is installed on the current system. If it is, ISMOUSE() returns a logical true, even if you use SET MOUSE OFF. If a mouse driver has not been installed, ISMOUSE() returns a logical false.

### Example

The following example uses ISMOUSE() to detect if a mouse is enabled or not, then displays the appropriate message.

```
IF ISMOUSE()  
  @ 10,10 SAY "The mouse is enabled"  
ELSE  
  @ 10,10 SAY "The mouse is disabled"  
ENDIF
```

### See Also

MCOL(), MROW(), ON MOUSE, SET MOUSE, SET()

---

## 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([[<.mdx filename>], <expN> [, <alias>]])

### Usage

If the optional .mdx filename is specified, <expN> refers to that file. If the specified file does not exist, KEY() returns an error message.

If no .mdx filename is specified, KEY() interprets <expN> with reference to all open indexes in the work area and checks .ndx file expressions first. KEY() next checks production .mdx tags and then non-production .mdx tags.

## KEY() KEYMATCH()

The optional alias parameter identifies a work area and can be a numeric expression from 1 to 40, a character expression from A through J, or a work area name. A work area name can be either a database filename or an alias name specified with the USE command. If you do not specify an alias, KEY() operates on the current work area. If the specified alias is not valid, KEY() returns an error message.

If you do not specify an index file with <expN>, KEY() returns the current controlling key expression.

### Example

This user-defined function, which assumes that a production .mdx file is active, takes the tag name as a parameter and returns a logical true (.T.) if the key expression is the same as the tag name:

```
FUNCTION TagSame
PARAMETER pn_tagname
PRIVATE lc_tagno
lc_tagno = TAGNO(pn_tagname)
RETURN TAG(lc_tagno) = UPPER(KEY(lc_tagno))
```

### See Also

DESCENDING(), FOR(), MDX(), NDX(), ORDER(), SET(), TAG(), TAGCOUNT(), TAGNO(), UNIQUE()

---

## KEYMATCH()

KEYMATCH() returns a logical true (.T.) or false (.F.) to indicate whether or not a specified expression is found in a specified index.

### Syntax

KEYMATCH (<exp>[,<index number>[,<exp work area>]])

where

<index number> = [<.mdx name>,<tag number>

or

<.ndx number>



### NOTES

1. <.ndx number> refers to an NDX file's position in the list of indexes. To find its position in the list, use DISPLAY STATUS or TAGNO().

2. To use <exp work area>, you must also specify <index number>. (Since both arguments can be numerical, using <exp work area> without <index number> would be ambiguous.)

## Usage

Use KEYMATCH() to determine if a specified expression is found in a particular index. A primary use of KEYMATCH() is to check for duplicate values during APPEND.

During APPEND, you can use KEYMATCH() with a VALID command to check if a matching index key already exists in the database. Since a new record's index key is added to an index after APPEND has been completed, using KEYMATCH() during APPEND tells if the new record will have the same index key as an existing record. If KEYMATCH() returns a logical true, you can take steps to restrict the user from adding the new record.

KEYMATCH() looks only in the specified index tag. It ignores the settings for SET DELETED, SET FILTER, and SET KEY, ensuring the data integrity of the database file even when you work with a subset of the database file's records. KEYMATCH() is useful when you want to check the index keys without changing the active index or moving the record pointer.

If you specify only the expression to match (<exp>), KEYMATCH() searches the current master index for an index key with the same value. If a matching index key is found, KEYMATCH() returns a logical true. Otherwise, it returns a logical false.

## Example

```
DEFINE WINDOW Dups FROM 12,10 TO 16,70
USE Employee
INDEX ON Lastname+Firstname TAG Lastfirst
CLEAR
m->lastname = SPACE(15)
m->firstname = SPACE(10)
@ 2,3 SAY "Lastname: " GET m->lastname
@ 3,3 SAY "Firstname: " GET m->firstname
.
.
.
```

## KEYMATCH() LASTKEY()

```
READ
IF READKEY() >255
  keyvalue = m->lastname + m->firstname
  keynum = TAGNO("Lastfirst")
  IF KEYMATCH(keyvalue,keynum)
    DO DupProc WITH m->lastname, m->firstname
  ELSE
    APPEND BLANK
    REPLACE lastname WITH m->lastname
    REPLACE firstname WITH m->firstname
  ENDIF
ENDIF
RETURN

PROCEDURE DupProc
PARAMETERS lname, fname
ACTIVATE WINDOW Dups
@ 2,2 SAY "An entry already exists for" +
  TRIM(fname) + " "+TRIM(lname)+". "
WAIT " (E)dit (R)eplace (C)ancel?" TO dupresult
.
.
.
RETURN
```

### See Also

FIND, FOUND(), INDEX, LOOKUP(), SEEK, SEEK()

---

## LASTKEY()

LASTKEY() returns the decimal ASCII value of the last key pressed to exit a full-screen command.

### Syntax

LASTKEY()

### Usage

LASTKEY() 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).

If the last action received during a keyboard wait state is a mouse click, LASTKEY() returns -100.



### 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)=Ct1-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.

### Example

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>/<memo field 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 is blank.

### 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, but it counts all carriage return line feeds as characters. The character count is increased by the number of lines in the memo field.

```
. ? LEN(Clien_hist)
      366
```

**See Also**

LTRIM(), RTRIM(), TRIM()

---

**LIKE()**

LIKE() is a programming function used for wildcard comparisons. It is also used to support the predicate in the set-oriented SQL language.

**Syntax**

LIKE (&lt;pattern&gt;, &lt;expC&gt;)

**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. Spaces cannot be included in the compared strings.

**Examples**

The LIKE() function lets you use wildcard symbols anywhere in a string, and use them as often as needed.

## LIKE() LINENO()

```
. ? 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 number of the command line that is currently executing 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.

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, ON ERROR, PROGRAM()

---

## 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(<expN>)

- expN=0 returns the time when the lock was placed.
- expN=1 returns the date when the lock was placed.
- expN=2 returns the log-in name of the user who locked the record or file.
- expN=3 returns the time of the last update or lock.
- expN=4 returns the date of the last update or lock.
- expN=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.

## 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()

LOG() returns the natural logarithm of a specified number.

### Syntax

LOG(<expN>)

### Usage

The natural logarithm has a base of the mathematical constant 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 real number 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. Raising e to 1 returns 2.71828.

```
. 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)+LOG(y)
      2.08
. ? EXP(2.08)
      8.00
```

### See Also

EXP(), LOG10()

## LOG10()

LOG10() returns the common log to the base 10 of a specified number.

### Syntax

LOG10(<expN>)

### Usage

LOG10() returns the value of the exponent in the following equation:

$$y = 10^x$$

Where y is the numeric expression used by the LOG10() function. This must be a positive real number for the value of <expN>. LOG10() returns the value of x, 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() or SUBSTR() functions are first called 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(), SUBSTR(), UPPER()

---

## LTRIM()

The LTRIM() function removes leading blanks from a character string.

### Syntax

LTRIM(<expC>)

### Usage

Use this function to remove leading blanks.

### Example

To remove leading blanks from a character string formed from a number:

```
. ? STR(11.95,8,2)
    11.95
. ? LTRIM(STR(11.95,8,2))
    11.95
```

### See Also

LEFT(), RIGHT(), RTRIM(), STR(), SUBSTR()

---

## LUPDATE()

LUPDATE() returns the date of the last update of the specified database file.

### Syntax

LUPDATE([<alias>])

### Usage

LUPDATE() displays the date of the last update of the active database file. If no database file is in USE, LUPDATE() returns a blank date.

The optional alias parameter identifies a work area and can be a numeric expression from 1 to 40, a character expression from A through J, an indirect filename reference, or a work area name. A work area name can be either a database filename or an alias name specified with the USE command. If you do not specify an alias, LUPDATE() operates in the current work area.

LUPDATE() returns the date in the form mm/dd/yy. This format can be modified with the SET CENTURY, SET DATE, and SET MARK commands.

## LUPDATE() MAX()

### Example

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(), MDX(), NDX(), RECCOUNT(), RECSIZE(), SET CENTURY, SET DATE, SET MARK

---

## 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. Character expressions are compared according to their ASCII values, reading from left to right.

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()

---

## MCOL()

MCOL() returns the column position of the mouse pointer on the screen.

### Syntax

MCOL()

### Usage

Use this function to determine the column position of the mouse pointer on the screen. Like the @ SAY/GET command, MCOL() reads the first visible column on the screen as column 0.

MCOL() returns the column position only if a mouse driver is installed and the mouse cursor is enabled. If the mouse drive isn't installed, or you use SET MOUSE OFF, MCOL() returns 0.

### Example

In the following example, dBASE displays the coordinates of the last mouse click in a display box unless the user clicks on a Quit button.

## MCOL() MDX()

```
SET TALK OFF
IF ISMOUSE()
  *- Set up the display window to display coordinates of last mouse click
  CLEAR                                && background color
  @ 3, 26 TO 8, 54 DOUBLE
  @ 4, 29 SAY "Last mouse click at:"
  @ 5, 30 SAY "MRow(): "
  @ 6, 30 SAY "MCol(): "
  @ 10, 36 SAY " Quit " COLOR w+/g
  !Quit = .F.
  DO WHILE .NOT. !Quit
    SET CONSOLE OFF
    WAIT                                && Wait for a mouse click
    SET CONSOLE ON
    nMRow = MRow()                      && Capture the mouse row and
    nMCol = MCol()                      && column as soon as possible
    nLKey = LASTKEY()
    IF nLKey = 81 .OR. nLKey = 113      && If 'Q' or 'q'
      !Quit = .T.
    ELSE
      *- Check for a click on the Quit button
      IF nMRow = 10 .AND. nMCol >= 36 .AND. nMCol <= 43
        !Quit = .T.                    && If so, signal an exit
      ENDIF
    ENDIF
    *- Display the row and column of the click
    @ 5, 38 SAY STR( nMRow, 2 )
    @ 6, 38 SAY STR( nMCol, 2 )
  ENDDO
ENDIF
SET TALK ON
CLEAR
```

For another example of how you can use MROW() and MCOL() in your programs, see Movewin.prg in the Samples directory.

### See Also

@, COL(), ISMOUSE(), MROW(), ON MOUSE, PCOL(), PROW(), ROW(), SET MOUSE

---

## MDX()

MDX() returns the filename for the .mdx file specified by the index order number.

### Syntax

MDX([<expN>[,<alias>]])

## Usage

MDX() returns the filename for the open, active .mdx file in the .mdx file list for the current work area. The optional <expN> parameter is the .mdx file position in the index file list specified by the SET INDEX TO <index file list>, SET ORDER TO, or USE ORDER commands. MDX() 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 you do not specify an index order number, MDX() returns the name of the .mdx file that contains the controlling index tag. If you do not specify an index order number and the controlling index is an .ndx file, MDX() returns a null string.

The optional alias parameter identifies a work area and can be a numeric expression from 1 to 40, a character expression from A through J, or a work area name. A work area name can be either a database filename or an alias name specified with the USE command. If you do not specify an alias, MDX() operates in the current work area.

MDX() honors the SET FULLPATH setting.

## See Also

FOR(), KEY(), NDX(), ORDER(), SET FULLPATH, SET(), TAG(), TAGCOUNT(), TAGNO()

---

## 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()

MEMORY() returns the amount of memory, in kilobytes, that is available in or allocated to various memory regions.

### Syntax

MEMORY([<expN>])

### Usage

Use MEMORY() to get information about your system's 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.

You can also use MEMORY() to determine if dBASE's Virtual Memory Manager (VMM) is running and how much memory it is managing. For more information about VMM, read the "Optimizing dBASE IV" chapter in *Programming in dBASE IV*.

The following table shows the valid values (0 to 7) for MEMORY(), and describes the result that each returns.

Value	Returns
0	Approximate amount of available memory. This is the sum of the values returned by MEMORY(3) and MEMORY(6).
1	Amount of available memory in the heap. A portion of this memory is available to dBASE.
2	Amount of low DOS memory (in the 640K region) that is available for loading bin files and running programs in DOS.
3	Amount of available memory that dBASE can allocate to various processes. This is virtual memory when running under VMM, or extended memory if dBASE isn't using VMM. VMM is disabled if you run dBASE under Windows or O/S2.
4	Total amount of memory that VMM has allocated for dBASE to use. This value is constant once dBASE is loaded. If dBASE isn't using VMM, MEMORY(4) returns 0.
5	Amount of extended memory VMM is managing. At start-up, VMM dynamically determines the amount of memory it can use <i>unless</i> you specify a different amount through the MAXMEM parameter in the dBASE.VMC file, or the DOS16M environmental setting. If dBASE isn't using VMM, MEMORY(5) returns 0.

## MEMORY() MENU()

- 6 Amount of memory allocated to the dBASE buffer manager that can be used by other dBASE processes as needed. For more information about the buffer manager, read the “Optimizing dBASE IV” chapter in *Programming In dBASE IV*.
  - 7 Size of the swap file, if VMM created one. VMM creates a swap file only if your system has insufficient extended memory to run dBASE IV. If VMM doesn't create a swap file, or if dBASE isn't using VMM, MEMORY(7) returns 0.
- 

MEMORY() and MEMORY(0) return the same result.

### Example

```
. ? MEMORY(6)  
1179
```

### See Also

DISKSPACE(), GETENV()

---

## 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 Help** 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**

See Appendix G for a list of all error messages generated by dBASE IV in single-user operation.

**See Also**

CERROR(), ERROR(), ON ERROR

---

**MIN()**

The MIN() function returns the smaller value of two numeric, date, or character expressions.

**Syntax**

MIN(&lt;expression1&gt;,&lt;expression2&gt;)

**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. Character expressions are compared according to their ASCII values, reading from left to right.

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 contains the smaller value:

```
. 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:

$$\langle \text{expN1} \rangle - \text{FLOOR}(\langle \text{expN1} \rangle / \langle \text{expN2} \rangle) * \langle \text{expN2} \rangle$$

where FLOOR is a mathematical function that returns the greatest integer less than or equal to its argument.

## MOD() MONTH()

### 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 05/15/92:

```
. ? 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()

## **MROW()**

MROW() returns the row position of the mouse pointer on the screen.

**Syntax**

MROW()

**Usage**

Use this function to determine the row position of the mouse pointer on the screen. Like the @ SAY/GET command, MROW() reads the first visible row on the screen as row 0.

MROW() returns the row position only if a mouse driver is installed and the mouse cursor is enabled. If the mouse driver isn't installed, or you use SET MOUSE OFF, MROW() returns 0.

**Example**

In the following example, dBASE displays the coordinates of the last mouse click in a display box unless the user clicks on a Quit button.

## MROW() NDX()

```
SET TALK OFF
IF ISMOUSE()
  *- Set up the display window to display coordinates of last mouse click
  CLEAR                                && background color
  @ 3, 26 TO 8, 54 DOUBLE
  @ 4, 29 SAY "Last mouse click at:"
  @ 5, 30 SAY "MRow(): "
  @ 6, 30 SAY "MCol(): "
  @ 10, 36 SAY " Quit " COLOR w+/g
  !Quit = .F.
  DO WHILE .NOT. !Quit
    SET CONSOLE OFF
    WAIT                                && Wait for a mouse click
    SET CONSOLE ON
    nMRow = MRow()                      && Capture the mouse row and
    nMCol = MCol()                      && column as soon as possible
    nLKey = LASTKEY()
    IF nLKey = 81 .OR. nLKey = 113      && If 'Q' or 'q'
      !Quit = .T.
    ELSE
      *- Check for a click on the Quit button
      IF nMRow = 10 .AND. nMCol >= 36 .AND. nMCol <= 43
        !Quit = .T.                    && If so, signal an exit
      ENDIF
    ENDIF
    *- Display the row and column of the click
    @ 5, 38 SAY STR( nMRow, 2 )
    @ 6, 38 SAY STR( nMCol, 2 )
  ENDDO
ENDIF
SET TALK ON
CLEAR
```

For another example of how you can use MROW() and MCOL() in your programs, see MOVEWIN.PRG in the SAMPLES directory.

### See Also

@, COL(), ISMOUSE(), MCOL(), ON MOUSE, PCOL(), PROW(), ROW(), SET MOUSE

---

## NDX()

NDX() returns the name of an .ndx file.

### Syntax

NDX ([<expN>[,<alias>]])



### Usage

NDX() allows programs to manipulate open index files without knowing the names of the files.

The optional parameter <expN> is the position of an open .ndx file in the index file list specified by SET INDEX TO <index file list> or USE INDEX <index file list>.

NDX() returns a null string if there is no index associated with <expN>.

If you do not specify an index position, NDX() returns the name of the current controlling .ndx file or a null string if the controlling index is associated with an .mdx file.

The optional alias parameter identifies a work area and can be a numeric expression from 1 to 40, a character expression from A through J, an indirect filename reference, or a work area name. A work area name can be either a database filename or an alias name specified with the USE command. If you do not specify an alias, NDX() operates in the current work area.

If the specified alias does not exist, dBASE IV returns the error message **ALIAS not found**.

NDX() honors the SET FULLPATH setting.

### Example

```
. USE Cust IN 2 INDEX Cus_name ALIAS Customer
Master index: CUS_NAME
. SELECT 3
. ? NDX(1,"Customer")
D:CUS_NAME.NDX
```

### See Also

ALIAS(), DBF(), FIELD(), MDX(), ORDER(), RECCOUNT(), RECSIZE(), SET FULLPATH, SET INDEX, SET(), SET ORDER

## NETWORK()

NETWORK() returns a logical true (.T.) if dBASE IV is running on a network or a logical false (.F.) if dBASE IV is not running on a network.

### Syntax

NETWORK()

**Usage**

This function is used in DOS programs to determine the system environment in which the program is running and to branch accordingly.

**See Also**

GETENV(), OS()

---

## **ORDER()**

ORDER() returns the name of the primary order index file or master .mdx tag.

**Syntax**

ORDER([<alias>])

**Usage**

ORDER() returns the name of the controlling index for the active database file or the database file specified by the alias name.

The optional alias parameter identifies a work area and can be a numeric expression from 1 to 40, a character expression from A through J, or a work area name. A work area name can be either a database filename or an alias name specified with the USE command. If you do not specify an alias name, ORDER() operates in the current work area.

ORDER() returns the root portion of the .ndx filename or the .mdx tagname in uppercase. If there is no active master index for the specified database, ORDER() returns a null string.

**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(), SET ORDER, TAG(), USE

---

## OS()

OS() 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. OS() returns the operating system name, release, and version number, each separated by a space.

### Examples

```
. ? OS()  
DOS 3.31
```

```
. ? OS()  
UNIX Xenix Sys_V 2.3.2
```

```
. ? OS()  
UNIX SunOS Sun 4.0
```

### 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  $\emptyset$  key and that option pad becomes the most recently selected pad. The PAD() function returns the name of that pad.

## PAD() PADPROMPT()

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()

---

## PADPROMPT()

PADPROMPT() returns the text that appears in a particular pad of a specific menu.

### Syntax

```
PADPROMPT(<expC1> [, <expC2>])
```

### Usage

Use PADPROMPT() to find out a pad's prompt text. You must specify the pad name (<expC1>), and can optionally specify the name of a menu (<expC2>). If you omit the menu name, dBASE uses the active menu.

PADPROMPT() requires that the menu and pad already be defined in the following way:

```
DEFINE MENU <menu name>
```

```
DEFINE PAD <pad name> OF <menu name> PROMPT <expC2>
```

In the previous example, PADPROMPT(<expC1>,<expC2>) returns the value of <expC2>.

### Example

```
. ? PADPROMPT("Pad1","Reports")    && Returns text in Pad1 of Reports  
PRINT
```

**See Also**

ACTIVATE MENU, DEFINE MENU, DEFINE PAD, MENU(), ON EXIT MENU, ON EXIT PAD, ON MENU, ON PAD, ON SELECTION MENU, ON SELECTION PAD, PAD(), POPUP(), PROMPT()

---

## PAYMENT()

PAYMENT() returns 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 a numeric expression for the principal balance of the loan. It can be positive or negative.

<rate> is the effective interest rate per period, expressed as a positive decimal number. If you are calculating a monthly payment using an annual interest rate, divide the rate by 12. For example, .015 represents an 18% annual interest rate.

<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**

```
INPUT "Enter the principal balance: " TO Principal
INPUT "Enter the annual interest rate: " TO Arate
Mrate=(Arate/10)/12
INPUT "Enter the number of payments: " TO Periods
Pmnt=PAYMENT(Principal,Mrate,Periods)
?"The monthly payment is",Pmnt
?"The total interest is", (Pmnt*Periods)-Principal
```

**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. (If SET DEVICE is not set to PRINTER, the second statement corresponds to the COL() function.)

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

---

## **PCOUNT()**

PCOUNT() returns the number of parameters passed to a procedure or user-defined function.

**Syntax**

PCOUNT()

**Usage**

Since version 1.5, dBASE IV does not return an error when the number of parameters passed to a procedure or user-defined function does not agree with the PARAMETER statement in the procedure or function.

dBASE IV ignores parameters in a DO WITH command that exceed the number specified in the PARAMETERS statement. If the DO WITH command passes fewer parameters to the procedure or user-defined function than are declared in the PARAMETERS statement, the excess parameters in the procedure or function are set to logical false (.F.).

Use PCOUNT() to determine the number of parameters passed to the procedure or user-defined function.

**Example**

This example shows the output of the following procedure:

```
PROCEDURE MyProg
PARAMETERS pc_1st, pc_2nd
?"PCOUNT() =",PCOUNT()
?"pc_1st =",pc_1st
?"pc_2nd =",pc_2nd
RETURN
```

```
. DO MyProg WITH "Hello"
PCOUNT() = 1
pc_1st = Hello
pc_2nd = .F.
```

**See Also**

DO, DEBUG, FUNCTION, PARAMETERS, PROCEDURE

---

**PI()**

The PI() function returns an approximation of  $\pi$ , the constant for the ratio of the circumference to the diameter of a circle.

**Syntax**

PI()

## PI() POPUP()

### Usage

The constant `PI` is used in mathematical and engineering calculations. `PI()` returns a type F number for this constant.

Use `SET DECIMALS` to control the number of decimal places displayed for the returned value of `PI()`.

### Examples

```
. ? PI()
3.14
```

Calculate the area of a unit circle, that is, a circle with a radius of 1. The area always equals  $\pi$ , and the units of measure are the same as the units used to measure the diameter such as inches, kilometers, or miles.

```
. Rad = 1
. Area = PI() * (Rad^2)
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()

PRINTSTATUS() 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** *Under DOS, only directly hard-wired printers will return an accurate PRINTSTATUS(). DOS network printers or those connected by an intervening device may not return an accurate response.*

### Example

The following code checks the status of the DOS 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
  ENDF
  IF Mready = "P"
    REPORT FORM Employee TO PRINT
  ENDF
  EXIT
ENDDO
```

### See Also

SET DEVICE, SET PRINTER

---

## **PROGRAM()**

PROGRAM() returns the name of the currently executing program, procedure, or user-defined function.

### **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 PROGRAM() 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 PROGRAM() from the dot prompt when there is no suspended program, it returns a null string.

An extension is not included in the returned program name.

If a procedure was executing, PROGRAM() returns the procedure name, not the program filename.

PROGRAM() 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()

## PROW() PV()

### 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()

PV() 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 per period. 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**

To determine the value of five payments of \$1.00 at 6% interest compounded annually:

```
. ? 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
```

## RAT()

RAT(), a variant of AT(), returns a number that shows the starting position of a character string (substring) within a larger string or memo field. RAT() starts the search from the *last* character of the string or memo field being searched. (AT() starts the search from the first character.)

### Syntax

RAT(<expC1>,<expC2>[,<expN>])

or

RAT(<expC1>,<memo field name>[,<expN>])


### Usage

Use the RAT() function to find the starting position of a character string within another string or memo field. The search is case-sensitive and starts, one character at a time, from the *last* character of the string or memo field being searched.

Specify the substring (<expC1>) you want to find, and the string or memo field (<expC2>) you want to search. You have the option of specifying the nth occurrence of the substring (<expN>) within the source string. If you omit this argument, RAT() returns the starting position of the *first* occurrence of the substring, after searching from the end of the source string.

Even though the search starts from the *right* of the source string, RAT() reads the characters from the *left* of the source string to determine the substring's starting position.

If the substring is not found, or is longer than the source string or memo field, RAT() returns a value of zero.

 **NOTE** *If you are searching a memo field, RAT() limits its search to approximately the first 64K of data.*

### Examples

```
. ? RAT("ss", "Mississippi")
6
. ? RAT("ss", "Mississippi", 2)
3
```

### See Also

AT(), LEFT(), RIGHT(), STUFF(), SUBSTR()

## READKEY()

READKEY() 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

Use READKEY() to determine what action your program should take after the user exits 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
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

(continued)

Table 4-3 READKEY() coding (continued)

34	290	<b>Ctrl-PgUp</b>	zoom out
35	291	<b>Ctrl-PgDn</b>	zoom in
36	292	<b>F1</b>	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**

Calling RECCOUNT() is a quick method of obtaining the number of physical records in a file.

The optional alias parameter identifies a work area and can be a numeric expression from 1 to 40, a character expression from A through J, or a work area name. A work area name can be either a database filename or an alias name specified with the USE command. If you do not specify an alias, RECCOUNT() operates in the current work area.

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.

**Example**

To find the number of records in the Client database file:



```
. USE Client  
. ? RECCOUNT()  
8
```

**See Also**

DBF(), DISKSPACE(), RECSIZE()

---

**RECNO()**

RECNO() returns the current record number of the specified file.

**Syntax**

RECNO([&lt;alias&gt;])

**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 before the first record in the file (BOF() is a logical true), RECNO() is 1.

The optional alias parameter identifies a work area and can be a numeric expression from 1 to 40, a character expression from A through J, or a work area name. A work area name can be either a database filename or an alias name specified with the USE command. If you do not specify an alias, RECNO() operates in the current work area.

RECNO() considers *all* records, even if SET DELETED is ON or SET FILTER is in effect.

**Example**

To display the current record number:

## RECNO() RECSIZE()

```
. 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

RECCOUNT(), SET RELATION

---

## RECSIZE()

RECSIZE() returns the size of a record in the specified database file.

### Syntax

RECSIZE([<alias>])

### Usage

The optional alias parameter identifies a work area and can be a numeric expression from 1 to 40, a character expression from A through J, or a work area name. A work area name can be either a database filename or an alias name specified with the USE command. If you do not specify an alias, RECSIZE() operates in the current work area.

If no file is in use, RECSIZE() returns a zero.

### Example

```
. USE District ALIAS Areas
. ? RECSIZE()
      23
. SELECT 2
. ? RECSIZE("Areas")
      23
```

### See Also

DBF(), FSIZE(), 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 or a character string.

### 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 *****

. ? REPLICATE ("abc", 2)
  abcabc
```

---

## 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()

RLOCK() 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.

The optional alias parameter identifies a work area and can be a numeric expression from 1 to 40, a character expression from A through J, an indirect filename reference, or a work area name. A work area name can be either a database filename or an alias name specified with the USE command. If you use an alias without a list of record numbers, the current record in that database file is locked.

ROCK() 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 100.

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, RLOCK() returns a logical true (.T.). Otherwise, it returns a logical false (.F.) and no 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.

## ROLLBACK() ROUND()

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 as limited by the SET DECIMALS command. SET DECIMALS affects only the display, not the result of ROUND().

### 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(), VAL()

## ROW()

The ROW() function returns the row number of the current cursor position.

**Syntax**

ROW()

**Usage**

ROW() is useful for relative screen addressing. Use it to control cursor positioning from within a program.

**Example**

The following program example prints two statements on the same line:

```
@ 1,5 SAY "This is the first part"
@ row (), col()+1 SAY "This is the next part"
```

You will see both statements on the same line.

**See Also**

@, COL(), PCOL(), PROW()

## RTOD()

RTOD() converts radians to degrees.

**Syntax**

RTOD(<expN>)

### Usage

The expression <expN> is the size of an angle measured in radians. RTOD() returns the angle size in degrees.

### Example

Convert 3/2 radians to degrees:

```
. x = 3 * PI()/2
4.71
. ? RTOD(x)
270
```

### See Also

ACOS(), ASIN(), ATAN(), ATN2(), COS(), DTOR(), SIN(), TAN()

---

## RTRIM()

RTRIM() removes trailing blanks from a character string. This function is identical to TRIM().

### Syntax

RTRIM(<expC>) or TRIM(<expC>)

### Usage

Use RTRIM() to trim trailing blanks from fields containing character strings.

### 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, you could use a single expression that causes RTRIM() to run the character strings together:

```
. ? RTRIM(Lastname)+Firstname
WrightFred
```



These two examples trim and space the fields with single blanks:

```
. ? RTRIM(Firstname)+" "+RTRIM(lastname)+", "+City  
Fred Wright, New York  
. ? "Dear "+RTRIM(Firstname)+" "+RTRIM(Lastname)+", "  
Dear Fred Wright,
```

### See Also

LEFT(), LTRIM(), RIGHT(), SET SPACE

---

## RUN()

RUN() loads an external program for execution by the operating system. This function returns a completion code from the external program or the operating system's command interpreter.

### Syntax

RUN([<expL1>,<expC>[,<expL2>])

### Usage

<expC> is the name of a command or program to be executed by the operating system. You must include the file extension if <expC> is a batch file (<expL1> is false (.F.)) and the full path name if the program is not on the default drive and path. You may include command line arguments after the command or program name.

The optional argument <expL1> controls whether <expC> is passed directly to the operating system.

If <expL1> is true (.T.), RUN() passes <expC> directly to the operating system. For DOS, dBASE IV uses the DOS EXEC function call to load and execute the specified program. When <expL1> is true (.T.):

- RUN() returns the completion or error code directly from the called program.
- RUN() cannot execute commands internal to the command interpreter, programs that rely on environment variables (such as PATH), or batch files. See example section for a workaround to this.

If <expL1> is false (.F.) or not specified, RUN() attempts to load the operating system's command interpreter to process <expC>. In DOS, RUN() uses the COMSPEC environment variable to locate the command interpreter. When <expL1> is false (.F.), RUN() returns completion codes from the command interpreter.

## RUN()

The optional argument <expL2> controls whether dBASE IV frees memory by rolling itself out to a swap file before the external program is executed. dBASE IV uses "TMP" as the first three letters of the temporary filename for the swap file. The file's extension is .sdb.

If <expL2> is true (.T.), dBASE IV leaves at least 10K of free space on the hard disk for use by the external program when it writes the swap file. The minimum amount of dBASE IV left in memory is approximately 10K. If writing dBASE IV from memory to disk would leave less than 10K of free disk space available, RUN() releases the dBASE IV overlay information from memory and the swap file is not created. When RUN() completes, the required overlays are restored to memory.

If <expL2> is false (.F.), RUN() loads <expC> for execution in the amount of memory available while dBASE IV is resident.

When <expL1> and <expL2> are false (.F.) or not specified, your computer must have enough available memory to load the command interpreter and the program that it will execute, otherwise dBASE IV returns the error message **Program too big to fit in memory**.

Do not execute programs that terminate and stay resident in memory (TSRs).

RUN() returns a numeric argument whose range is operating system dependent. In DOS, the status code may be between 0 and 255.

### Examples

This example loads the DOS external Chkdsk.com program for direct execution by the operating system and RUN() returns DOS completion code 0 to indicate that the program completed successfully:

```
.? RUN(.T., "C:\DOS\Chkdsk.com c:")

42496000 bytes total disk space
  55296 bytes in 2 hidden files
 110592 bytes in 52 directories
10803200 bytes in 296 user files
31526912 bytes available on disk

655360 bytes total memory
238032 bytes free
0
```

There is a way to execute batch files when expL1 is true. You need to load a second version of COMMAND.COM then execute your batch file:

```
. ? RUN(.T., "C:\COMMAND.COM /C MYBAT")
```

You must have enough memory to load the command interpreter and the batch file to do this.

This example loads the program Prog2.exe for execution by the operating system and rolls dBASE IV out of memory into a temporary swap file on the hard disk:

```
..? RUN(.T., "C:\BIN\Prog2.exe", .T.)
```

### See Also

CALL, CALL(), GETENV(), LOAD, RUN/!

## SEEK()

SEEK() searches an indexed database file for a specified expression.

### Syntax

```
SEEK(<exp> [, <alias>])
```

### Usage

SEEK() evaluates the expression specified by <exp> and attempts to find its value in the master index of the specified database file. It returns a logical true (.T.) if the index key is found, and a logical false (.F.) if it is not found.

The optional alias parameter identifies a work area and can be a numeric expression from 1 to 40, a character expression from A through J, an indirect filename reference for an open, indexed database file, or a work area name. A work area name can be either a database filename or an alias name specified with the USE command. If you do not specify an alias, SEEK() operates in the current work area.

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, the SEEK command and FOUND() function can be combined to execute a specific action when the keyword is found. For example:

```
SEEK M_id
IF FOUND()
    * - Do something
ENDIF
```

## SEEK()

Because the SEEK() function both moves the record pointer and returns a logical result, it can be used instead of a FIND/SEEK and FOUND() or EOF() combination (as shown above). 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()

SELECT() returns the number of an available work area or the work area number associated with a specified alias.

### Syntax

```
SELECT([<alias>])
```

### Usage

The optional alias parameter identifies a work area and can be a numeric expression from 1 to 40, a character expression from A through J, or a work area name. A work area name can be either a database filename or an alias name specified with the USE command. If an alias is specified, SELECT() returns the number of the work area associated with that alias.

If you do not specify an alias, SELECT() returns the number of the next available work area.

SELECT() returns a number between 1 and 40 representing the work area. This function returns 0 when:

- there are no more work areas available
- the specified alias is not in use (this a good way to test whether the alias exists in any work area)

If the specified alias is a numeric expression outside the range of 1 to 40, SELECT() returns an error message.

Note that there may be insufficient memory to allocate a valid work area number returned by SELECT().

### Examples

```
. ? SELECT()
11
```

To open a file in an unused area:

```
. Filename = "Client"
. USE (Filename) IN SELECT() ALIAS Customers
. ?SELECT("Customers")
2
```

### See Also

ALIAS()

## SET()

SET() returns the status of various SET commands.

### 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
ALTERNATE	ON/OFF
ATTRIBUTES	SET COLOR values*
AUTOSAVE	ON/OFF
BELL	ON/OFF
BLOCKSIZE	Integer
BORDER	<border type>
CARRY	ON/OFF
CATALOG	ON/OFF
CENTURY	ON/OFF
CLOCK	ON/OFF
COLOR	ON/OFF
CONFIRM	ON/OFF
CONSOLE	ON/OFF
CURSOR	ON/OFF
DATE	<date format>
DEBUG	ON/OFF
DECIMALS	Integer
DEFAULT	<drive>
DELETED	ON/OFF
DELIMITERS	ON/OFF

(continued)

Table 4-4 SET() keywords and return values (continued)

<b>SET Keyword</b>	<b>Value Returned</b>
DESIGN	ON/OFF
DEVELOPMENT	ON/OFF
DEVICE	<device>
DIRECTORY	<path>
DISPLAY	<display mode>
ECHO	ON/OFF
ENCRYPTION	ON/OFF
ESCAPE	ON/OFF
EXACT	ON/OFF
EXCLUSIVE	ON/OFF
FIELDS	ON/OFF
FILTER	<filter expression>**
FORMAT	<format filename>
FULLPATH	ON/OFF
HEADINGS	ON/OFF
HELP	ON/OFF
HISTORY	ON/OFF
HOURS	12 or 24
IBLOCK	Integer
INDEX	<index name>
INSTRUCT	ON/OFF
INTENSITY	ON/OFF
LDCHECK	ON/OFF
LIBRARY	<filename>
LOCK	ON/OFF
MARGIN	Integer
MARK	<date delimiter>
MBLOCK	Integer
MEMOWIDTH	Integer

(continued)

Table 4-4 SET() keywords and return values (continued)

<b>SET Keyword</b>	<b>Value Returned</b>
MOUSE	ON/OFF
NEAR	ON/OFF
ORDER	<.ndx filename>/<tag>
ODOMETER	Integer
PATH	<path>**
PAUSE	ON/OFF
POINT	<expC>
PRECISION	Integer
PRINTER	ON/OFF
PROCEDURE	<filename>
REFRESH	Integer
RELATION	<expression> INTO <alias>...**
REPROCESS	Integer
SAFETY	ON/OFF
SCOREBOARD	ON/OFF
SEPARATOR	<separator char>
SKIP	<alias>,<alias>...**
SPACE	ON/OFF
SQL	ON/OFF
STATUS	ON/OFF
STEP	ON/OFF
TALK	ON/OFF
TITLE	ON/OFF
TRAP	ON/OFF
TYPEAHEAD	Integer
UNIQUE	ON/OFF
VIEW	<filename>
WINDOW	<window name>

(continued)



Table 4-4 SET() keywords and return values (continued)

\*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.

\*\*SET("FILTER"), SET("PATH"), SET("RELATION") and SET("SKIP") return only the first 254 characters. All characters beyond that limit are truncated.

### 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 hold the status of certain settings:

```
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

---

## 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()

---

## SIN()

The SIN() function returns the trigonometric sine of an angle.

### Syntax

SIN(<expN>)

<expN> is a numeric expression representing the size of the angle in radians.

### Usage

Use this function to get the sine of an angle. No limits are placed on the argument. SIN() returns a type F number.

### Example

The value of sine varies between the limits of +1 and -1, passing through zero at 0 radians,  $\pi$  radians, and  $2\pi$  radians.

```
. ? SIN(PI()/2)
      1
. ? SIN(PI())
      0
. ? SIN(3*PI()/2)
     -1
. ? SIN(2*PI())
      0
```

The SET DECIMALS command determines 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:

## SOUNDEX()

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
. ? STR(x*10,5)      11.14      && no decimal places
  111
. ? STR(x*10,5,1)   111.4      && one decimal places
  111.4
. ? STR(x*10,5,2)   111.4      && two decimal places
  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:

## STUFF() SUBSTR()

```
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.

### 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

The following examples extract substrings from larger strings and from variables containing strings:

```
. ? 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()

TAG() returns the TAG name in a specified .mdx file.

### Syntax

TAG([[<.mdx filename>,<expN>[,<alias>]])

### Usage

This function returns a character string representing the name of an index file (.ndx) or .mdx tag for the index specified by the optional parameter <expN>. If the optional .mdx filename is specified, <expN> refers to that file. TAG() returns an error condition if the specified .mdx file or index do not exist.

If no index is specified, TAG() returns the name of the controlling index.

If no .mdx filename is specified, TAG() interprets <expN> with reference to all open indexes in the work area and checks .ndx file expressions first. TAG() next checks production .mdx tags and then non-production .mdx tags.

The optional alias parameter identifies a work area and can be a numeric expression from 1 to 40, a character expression from A through J, or a work area name. A work area name can be either a database filename or an alias name specified with the USE command. If you do not specify an alias, 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(), TAGCOUNT(), TAGNO()

---

## TAGCOUNT()

TAGCOUNT() returns the number of active indexes in a specified work area or .mdx file.

### Syntax

TAGCOUNT([[<.mdx filename>[,<alias>]])

## TAGCOUNT() TAGNO()

### Usage

If the .mdx filename is not specified, TAGCOUNT() returns the total number of indexes in the indicated work area (.ndx files are included).

The optional alias parameter identifies a work area and can be a numeric expression from 1 to 40, a character expression from A through J, or a work area name. A work area name can be either a database filename or an alias name specified with the USE command.

If you do not specify an alias, TAGCOUNT() returns the total number of indexes in the current work area (.ndx files are included).

TAGCOUNT() returns an error message if the specified .mdx filename is not active in the indicated work area.

### Examples

To determine the number of tags associated with the first .mdx file for the database file in the current work area:

```
. Indx=MDX(1)
. ? Indx
D:\STAFF\FRED\MORETAGS.MDX
. ? TAGCOUNT(Indx)
      2
```

For the first .mdx file in work area 3:

```
. ? MDX(1,3)
D:\DBASE\SAMPLES\CONTENTS.MDX
. ? TAGCOUNT("Contents.mdx",3)
      1
```

### See Also

KEY(), MDX(), TAG(), TAGNO()

---

## TAGNO()

TAGNO() returns the index number for the specified index.

### Syntax

TAGNO([<order name>[, <.mdx filename> [,<alias>]]])

## Usage

Use the optional parameter <order name> to specify the name of the index tag or .ndx file whose position you want. The optional parameter <mdx name> is the name of the .mdx file that contains the tag. If you do not specify an .mdx filename, TAGNO() interprets <order name> with reference to all open indexes in the work area, including .ndx files which are always at the top of the index list.

If you do not specify an index name, TAGNO() returns the position of the controlling index.

The optional alias parameter identifies a work area and can be a numeric expression from 1 to 40, a character expression from A through J, or a work area name. A work area name can be either a database filename or an alias name specified with the USE command. If you do not specify an alias, TAGNO() operates in the current work area.

TAGNO() returns a value for the specified index's sequence in the index list.

TAGNO() returns error conditions when:

- the specified index tag does not exist
- the specified .mdx file is not active in the indicated work area

The user can perform several actions that change the order of the indexes. For this reason, TAGNO() function calls should be made immediately before the index number is required.

## Examples

The first example assumes DISPLAY STATUS shows the following indexes in use:

```
. DISPLAY STATUS
Currently Selected Database:
Select area: 1, Database in Use: D:\DATA\STAFF.DBF Alias: STAFF
  Master Index file: D:\DATA-AMES.NDX
  Production MDX file: D:\DATA\STAFF.MDX
    Index Tag: LASTNAME Key: LASTNAME
    Memo file: D:\DATA\STAFF.DBT
```

```
. ? TAGNO("Lastname")
      2
. ? TAGNO("Lastname","Staff.mdx")
      1
```

This example finds the key expression for the master tag:

## TAGNO() TAN()

```
. SELECT 2
. ? MDX(1)
D:EMPLOYEE.MDX
. ? TAGNO(ORDER(), "Employee.mdx")
. ? KEY(7)
  lastname+firstname+initial
```

### See Also

KEY(), MDX(), ORDER(), SET("INDEX"), SET("ORDER"), TAG(), TAGCOUNT()

---

## 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  $\pi/2$  radians.

This function returns a type F number. The SET DECIMALS command determines the accuracy of the display.

### Example

```
. ? TAN(PI())
  0
```

### See Also

ACOS(), ASIN(), ATAN(), ATN2(), COS(), SET DECIMALS, SIN()

---

## TIME()

TIME() returns the system time as a character string in one of the following formats: HH:MM:SS or HH:MM:SS.hh. HH is the hour, MM is the minute, SS is the second, and hh is the hundredths of a second.

### Syntax

TIME([<exp>])

### Usage

Use TIME() to return the system time. If you specify a valid expression (<exp>), TIME() returns the system time in HH:MM:SS.hh format. If you use TIME() without an expression, it returns the system time in HH:MM:SS format.

To use TIME() in calculations, convert the value returned to a numeric value using SUBSTR() and VAL(). TIME() always returns the time in 24-hour format; it is not affected by SET HOURS TO [12/24].

### Example

```
. ? TIME()  
17:45:40  
. ? TIME("x")  
17:45:41.10  
. ? TIME(5)  
17:45:42.05
```

### See Also

DATE(), SET HOURS

---

## 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.

## TRANSFORM() TRIM()

### 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 expression and returns a single uppercase character that indicates whether the expression evaluates to a character, numeric, logical, memo, date, float, or undefined data type.

### 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 character expression 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 expression 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 logical field in Transact.dbf.
L
```

## TYPE() UNIQUE()

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.
```

A fast method of determining if a field name exists in a file is to use the TYPE() function. If TYPE() returns a "U", the file does not contain the field. This example shows that the Client.dbf database file contains a field named Lastname, but not a field named Vendor.

```
. ? TYPE("Client->Lastname")  
C  
. ? TYPE("Client->Vendor")  
U
```

---

## UNIQUE()

UNIQUE() returns a logical true (.T.) if the specified index was created with the UNIQUE keyword or with SET UNIQUE ON.

### Syntax

UNIQUE([[<mdx filename>], <expN> [, <alias>]])



### Usage

The UNIQUE() function returns a logical true (.T.) if the index number specified by the optional <expN> parameter was created with the UNIQUE keyword or the SET UNIQUE command ON. It returns a logical false (.F.) if the specified index is not unique.

If you do not specify an index number, UNIQUE() tests whether the controlling tag was created with the UNIQUE keyword.

The optional <.mdx filename> parameter specifies an .mdx filename.

The optional alias parameter identifies a work area and can be a numeric expression from 1 to 40, a character expression from A through J, or a work area name. A work area name can be either a database filename or an alias name specified with the USE command. If you do not specify an alias, UNIQUE() operates in the current work area.

UNIQUE() returns an error condition if the specified index or index filename does not exist.

### Example

```
. USE Employee
. INDEX ON Lastname TO Name1 UNIQUE
. ? TAGNO("Name1")
    3
. ? UNIQUE(3)
.T.
. INDEX ON Lastname TAG Name2 OF Moretags
. ? TAGNO("Moretags", "Name2")
    2
. ? UNIQUE(2)
.F.
```

### See Also

DESCENDING(), FOR(), INDEX, KEY(), MDX(), NDX(), ORDER(), SET UNIQUE, TAG(), TAGCOUNT(), TAGNO()

---

## UPPER()

The UPPER() function converts lowercase letters in a character string to uppercase letters. It does not work with memo fields unless the LEFT(), MLINE(), RIGHT(), or SUBSTR() functions are used first to extract a character string from the memo field.

### Syntax

UPPER(<expC>)

## UPPER()

### 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, USER() returns a null string.

### See Also

LIST/DISPLAY USER, 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.

## VAL() VARREAD()

### 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 Help** 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 Help** 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([expN])

### Usage

The VERSION() function returns the program version number. With a numeric argument, VERSION() returns the exact build number and build date.

This function is useful in applications that utilize version-specific features.

### 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
. DEFINE WINDOW Entry2 FROM 5,5 TO 20,20
. 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 05/15/91:

```
. ? YEAR(DATE())  
1991
```

To store the year from the system date to the memory variable:

```
. Myear = YEAR(DATE())  
1991  
  
. ? TYPE("Myear")
```

To increase the current system year by 50 and store the resulting year to the numeric memory variable Newyear:

```
. Newyear = YEAR(DATE()) + 50  
2041
```

**See Also**

DAY(), MONTH()

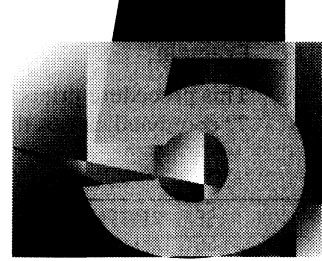




# **System Memory Variables**



# System Memory Variables



System memory variables control printed output. They are assigned default values by dBASE at start-up. They can be changed from the dot prompt or from programs. All system variables can be polled to return their current settings.

---

## **\_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**

\_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.). However, only the DEFINE BOX output can be inserted into streaming output. Drawing a box and placing text in it with the @...TO command is difficult.

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
_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
?
```

## See Also

`DEFINE BOX`, `SPACE()`

---

## `_indent`

`_indent` specifies the indentation of the first line of each new paragraph printed with the `?` command when `_wrap` is true (`.T.`).

### Syntax

`_indent = <expN>`

### Default

The default paragraph indentation is 0.


### Usage

The `_indent` system variable instructs dBASE IV to indent the first line of each new paragraph by the specified number of characters. The value of `_indent` can range from (`_lmargin`), which is the negative value of the `_lmargin` setting, to (`_rmargin - _lmargin - 1`). For example, if `_lmargin` is 10, `_indent` may be as low as -10, and the first line of each paragraph will output ten columns to the left of the rest of the paragraph. If `_rmargin` is 75, `_indent` may be as high as 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
_lwrap = .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 issues the form feed and takes over 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
- when the program reaches a PRINTJOB or ENDPRINTJOB and the `_peject` system memory variable causes an eject

dBASE IV uses the formula  $(\_plength - \text{MOD}(\text{PROW}(), \_plength))$  if the eject is not during streaming output mode. This occurs if you issue an EJECT command, or if you SET DEVICE TO PRINTER and force a page eject with the @ command.

Sending a CHR(12) to the printer always issues a form feed, even if `_padvance` is set to "LINEFEEDS".

If you're printing short pages, such as checks that are 20 lines long, you don't have to adjust the form length setting on the printer. Instead, set `_plength` to the length of your output and `_padvance` to "LINEFEEDS". dBASE IV then bypasses the printer form length setting, since it doesn't send a form feed.

### Options

"FORMFEED" — Advance paper a sheet at a time.

"LINEFEEDS" — Advance paper one line at a time.

### Example

In the following example, the `Chk_prin.prg` file (which is not provided on disk in your dBASE IV package) prints a check for every record in the `Debts.dbf` database file:

```
PRIVATE _plength, _padvance
_plength = 20
_padvance = "LINEFEEDS"
USE Debts
SCAN
DO Chk_prin          && Print a check.
    EJECT            && Advance to the top of the next check.
ENDSCAN
CLOSE DATABASE
```

### See Also

`_peject`, `_plength`, `_plineno`, EJECT, PROW()



---

## **\_pageno**

\_pageno determines or sets the current page number.

### **Syntax**

\_pageno = <expN>

### **Default**

The default page number is 1.


### **Usage**

\_pageno works on the data stream that is being output. This *streaming output* is produced by all commands that create output, except the @, @...TO, and EJECT commands.

The \_pageno system variable can both determine the current page number and set the page number to a specified value. Use it to print page numbers in a report or, when you are combining documents, to assign a suitable number to the first page of the second document. You can assign \_pageno an integer value ranging from 1 through 32,767.

dBASE IV keeps track of the current page number in your streaming output. It increments this variable as it goes to the next page of the streaming output, since the point where the pages break depends upon the actual text in the report. The \_pageno value depends upon the value of \_plineno and \_plength, which are discussed elsewhere in this chapter.

Don't confuse \_pageno with \_pbpage, the system variable that tells dBASE IV on what page to begin printing.

 **TIP** *\_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.*

## Examples

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.

 **TIP** *The three system variables `_pageno`, `_pbpage`, and `_pepage` should be consistent. For example, if `_pbpage` = 3, `_pepage` = 5, and `_pageno` = 10, dBASE IV won't print any pages, because the current page number (10) is outside the range of pages to be printed (3 to 5).*

*The value of `_pageno` should be less than or equal to the value of `_pepage`. For the example just given, `_pageno` should be less than or equal to 5 to print any pages. You can use this feature to print just part of a document by suitably setting `_pageno`, `_pbpage`, and `_pepage`.*

*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 altered changed text.
```

### See Also

`_plineno`, `_rmargin`, `LEN()`, `PCOL()`, `REPLICATE()`, `SET DEVICE`, `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 set `_pcopies` from the dot prompt or in programs. You need to run a program to see its effect, because it requires the `PRINTJOB/ENDPRINTJOB` command. In a program, place the `_pcopies` variable definition before the `PRINTJOB` command.



**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".*

**\_pcopies**  
**\_pdriver**

### 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.)

```
@ 10,20 SAY "Number of copies: " GET _pcopies PICTURE "99"  
READ  
PRINTJOB  
.      && <Printjob commands>  
.  
.  
ENDPRINTJOB
```

### See Also

@, PRINTJOB/ENDPRINTJOB

---

## **\_pdriver**

\_pdriver activates the desired printer driver or returns the name of the current driver.

### Syntax

\_pdriver = "<printer driver filename>"

### Default

When you first install dBASE IV, you assign a default printer driver. dBASE IV writes this default driver in your Config.db file. See *Getting Started with dBASE IV* for more information on your Config.db printer settings.

You can change the active printer driver by issuing \_pdriver = "<printer driver filename>" from the dot prompt.

The device that the driver prints to is the current default device, 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 filename, and may include a path. You don't need to type the file extension; dBASE IV assumes it to be .pr2. Postscript is an exception and you need to use the extension.

Fonts configured in Config.db are associated by setting the driver with \_pdriver statements.

dBASE IV supports the printer drivers listed in the on-disk printer drivers file. 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 on the printer selection menu available to you in DBSETUP. 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.



**TIP** *The ASCII printer driver prints ASCII characters without interpreting them for the printer. If characters from 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.pr3", 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.pr3" 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 = "Hplasl00.pr2"
CASE mprinter = "IBM Quietwriter"
  _pdriver = "Ibmquiet.pr2"
OTHERWISE
  _pdriver = "Generic.pr2"
ENDCASE
```

### See Also

[\\_ppitch](#), [\\_pquality](#), [?/?/?](#), [SET PRINTER](#)

The *Getting Started with dBASE IV* manual.

---

## **\_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. However, it can be defined from the dot prompt or in a program. It will take effect only when you run a program. 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 can send the mnemonic inside curly braces, the character ASCII value inside curly braces, or the character itself. The length of `<expC>` is limited to 254 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
```

or

```
_pecode = "!ESC|E"      && Also sends Esc E
```

### See Also

`_pscode`, `?/?` (`STYLE` option), `???`, `PRINTJOB/ENDPRINTJOB`

Your printer manual has additional information about printer control codes.

## **`_peject`**

`_peject` controls ejecting a sheet of paper before and 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`.

 **NOTE** *Many laser and PostScript printers require a form feed after the last page in order to print the last page.*

### 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 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**

`_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"
```

### **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.

The current pitch remains active until a new pitch is asked for.

### **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.

This variable may produce an error message if used with laser printers. It does not support laser printers.



 **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`

The printer drivers disk file 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 can send the mnemonic inside curly braces, the character ASCII value inside curly braces, or the character itself. The length of `<expC>` is limited to 254 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.

### Examples

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}"
```

This sets Compressed Mode on an Epson FX+ printer:

```
_pscode = CHR(15)
```

### See Also

`_pcode`, `?/??` (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.

\_pwait can be defined at the dot prompt or in programs. In programs, it should be defined before the PRINTJOB command.

### Example

To make printing pause between pages, set \_pwait to true (.T.):

```
. _pwait = .T.
```

**\_pwait**  
**\_rmargin**

To preview a file to the screen:

```
SET STATUS OFF
USE Employee
_pwait=.T.
SET PRINTER TO NUL
_plength=22
REPORT FORM EMPLOYEES TO PRINT
```

### See Also

\_plength, EJECT, EJECT PAGE

---

## **\_rmargin**

\_rmargin defines the paragraph right margin for the output of the ??? command when \_wrap is true (.T.).

### Syntax

\_rmargin = <expN>

### Default

The default right margin is 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.

### 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`

Chapter 2 of *Getting Started with dBASE IV*, 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`

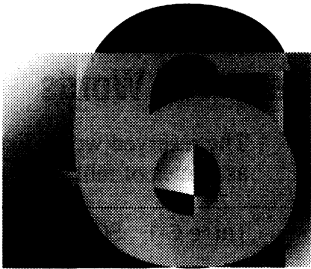




# SQL Commands



# 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.

### Written Syntax



### Graphic Form of Syntax

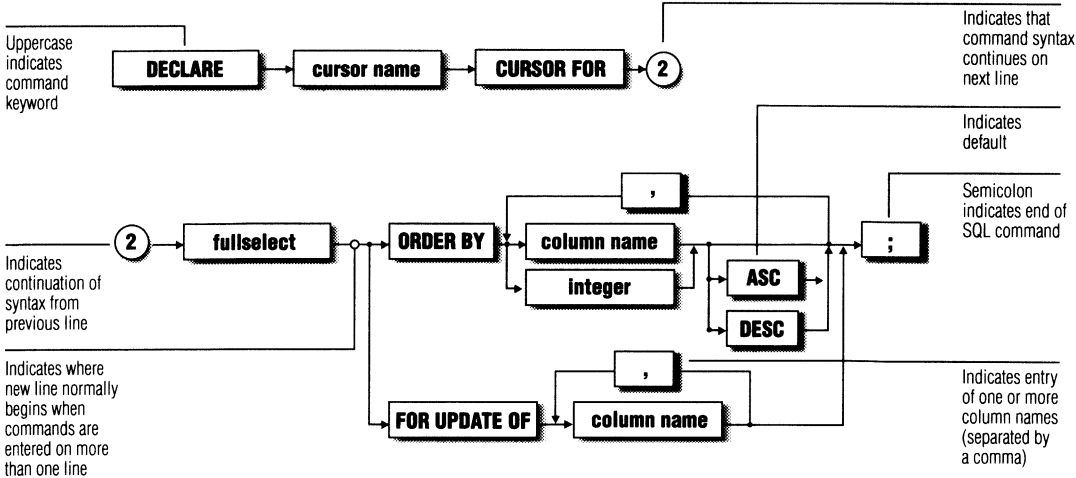



Figure 6-1 Command syntax

## 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	DECLARE	LOGICAL	SIGN
ACOS	DELETE	LOGIN	SIN
ADD	DELIMITED	LOGOUT	SMALLINT
ALL	DESC	LOWER	SOUNDEX
ALTER	DIF	LTRIM	SPACE
AND	DIFFERENCE	MAX	SQRT
ANY	DISTINCT	MDY	START
AS	DMY	MIN	STOP
ASC	DOW	MOD	STR
ASIN	DROP	MONTH	STUFF
AT	DTOC	NOT	SUBSTR
ATAN	DTOR	NUMERIC	SUM
ATN2	DTOS	OF	SYLK
AVG	EXISTS	ON	SYNONYM
BETWEEN	EXP	OPEN	TABLE
BLANK	FETCH	OPTION	TAN
BY	FIXED	OR	TEMP
CDOW	FLOAT	ORDER	TIME
CEILING	FLOOR	PAYMENT	TO
CHAR	FOR	PI	TRANSFORM
CHECK	FROM	PRIVILEGES	TRIM
CHR	FV	PUBLIC	TYPE
CLOSE	FW2	PV	UNION
CLUSTER	GRANT	RAND	UNIQUE
CMONTH	GROUP	REAL	UNLOAD
COS	HAVING	REPLICATE	UPDATE
COUNT	IN	REVOKE	UPPER
CREATE	INDEX	RIGHT	USER
CTOD	INSERT	ROLLBACK	USING
CURRENT	INT	ROUND	VAL
CURSOR	INTEGER	RPD	VALUES
DATA	INTO	RTOD	VIEW
DATABASE	KEEP	RTRIM	WHERE
DATE	LEFT	RUNSTATS	WITH
DAY	LEN	SAVE	WKS
DBASEII	LIKE	SDF	WORK
DBCHECK	LOAD	SELECT	YEAR
DBDEFINE	LOG	SET	
DECIMAL	LOG10	SHOW	

 **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/Start-up 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>...]);
```

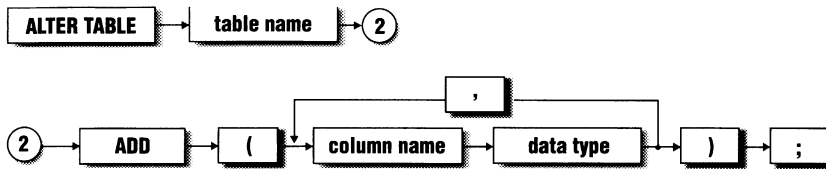


Figure 6-2 ALTER TABLE command

### 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, <i>mm\dd\yy</i> )
DECIMAL(x,y)	Fixed decimal point numbers
FLOAT(x,y)	Floating point numbers
Data Type	Description
INTEGER	11-digit integer values

(continued)



Table 6-2 SQL data types (*continued*)

<b>Data Type</b>	<b>Description</b>
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. 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.

**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

## CLOSE

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.

 **NOTE** *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>;
```

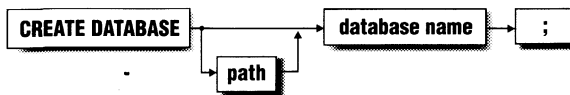


Figure 6-4 CREATE DATABASE command

### 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 with dBASE IV.

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. The full path is recorded in this table. If you change directories, dBASE IV will have trouble locating databases.

If the database directory specified using <path> does not already exist, a new directory is created in the directory specified by the path. Only the the subdirectory is created, SQL does not create the implied directories preceding the database subdirectory. 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.

**WARNING** *If you use CREATE DATABASE to define a database with the same name and path specification as your SQLHOME directory, you will get an error message.*

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]...]);
```

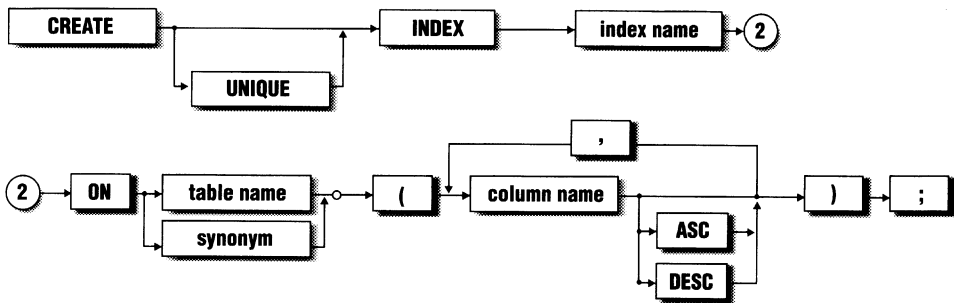


Figure 6-5 CREATE INDEX command

### 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.*

## CREATE INDEX CREATE SYNONYM

### 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 Inventory table, type:

```
SQL. CREATE INDEX Expense  
ON Inventory (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>;
```

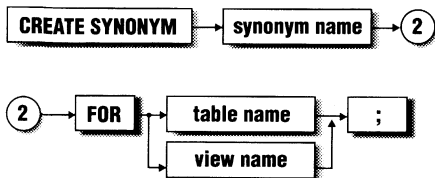


Figure 6-6 CREATE SYNONYM command



## 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 Synname = "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>...]);
```

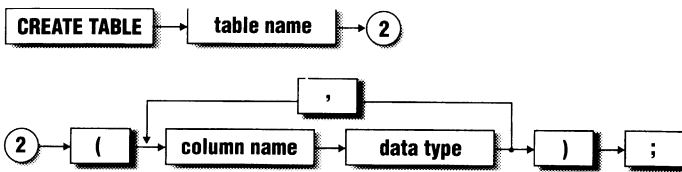


Figure 6-7 CREATE TABLE command

### 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

<b>Data Type</b>	<b>Description</b>
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 <i>x</i> total digits (including sign) and <i>y</i> decimal places (significant digits to the right of the decimal point). <i>x</i> may range from 1 to 19 and <i>y</i> 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 <i>x</i> total digits (including sign and decimal point) and <i>y</i> decimal places (significant digits to the right of the decimal point). <i>x</i> may range from 1 to 20 and <i>y</i> 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 <i>x</i> total digits (including sign and decimal point) and <i>y</i> decimal places (significant digits to the right of the decimal point). <i>x</i> may range from 1 to 20 and <i>y</i> may range from 0 to 18. The range of numbers you may store is $0.1 \cdot 10^{-307}$ to $0.9 \cdot 10^{+308}$ .  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 <i>n</i> characters. <i>n</i> 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 <i>mm/dd/yy</i> . 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];
```

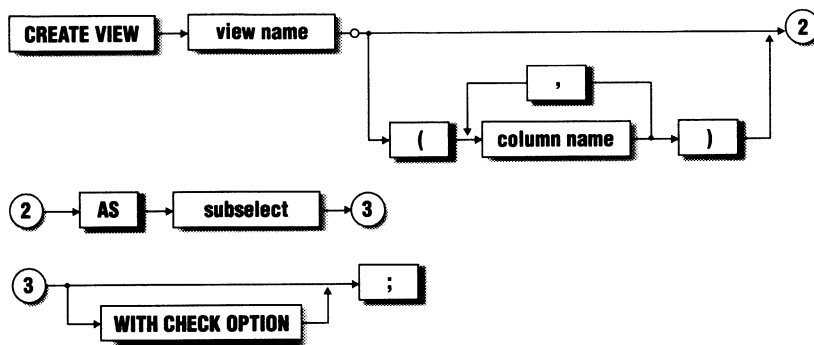


Figure 6-8 CREATE VIEW command

### 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>];

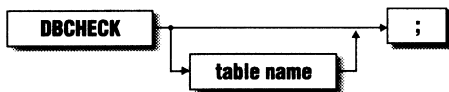


Figure 6-9 DBCHECK command

### 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-opened error.

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 DBDEFINING 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;
```

## DBDEFINE DECLARE CURSOR

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.

### Syntax

```
DECLARE <cursor name> CURSOR  
  FOR <SELECT command>  
  [FOR UPDATE OF <column list>/<ORDER BY clause>];
```

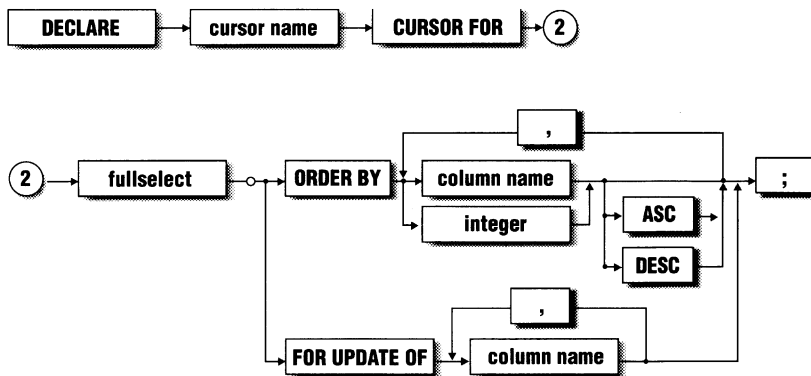


Figure 6-11 DECLARE CURSOR command

### 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.

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.

## DECLARE CURSOR

### 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>;
```

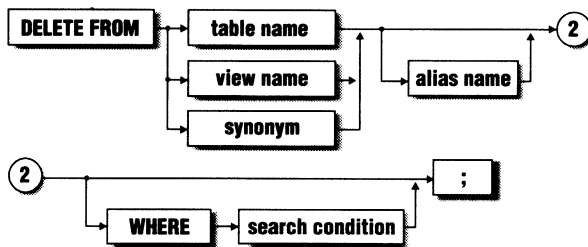


Figure 6-12 DELETE...WHERE <clause> form of command

```
DELETE (2)
  FROM <table name>
  WHERE CURRENT OF <cursor name>;
```

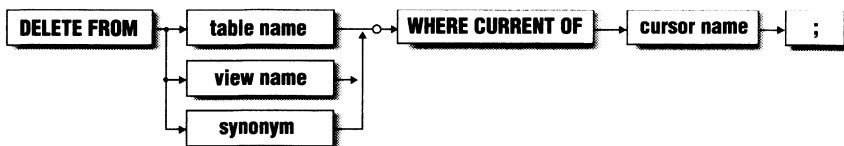



Figure 6-13 DELETE...WHERE CURRENT OF form of command


### 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 command. This is particularly important if others are simultaneously viewing the same data.

### 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 < 1091221871;
```

The system displays the message **4 row(s) deleted.**

## DELETE DROP DATABASE

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>;

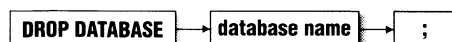


Figure 6-14 DROP DATABASE command

**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** *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>;

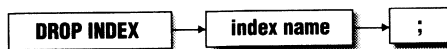


Figure 6-15 DROP INDEX command

### 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>;

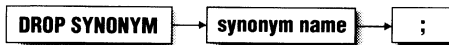


Figure 6-16 DROP SYNONYM command

### 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>;

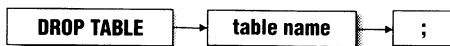


Figure 6-17 DROP TABLE command

## 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

## **FETCH**

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>;
```

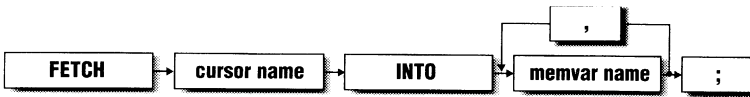


Figure 6-19 FETCH command

### **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
    ENDF
  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>)]
```



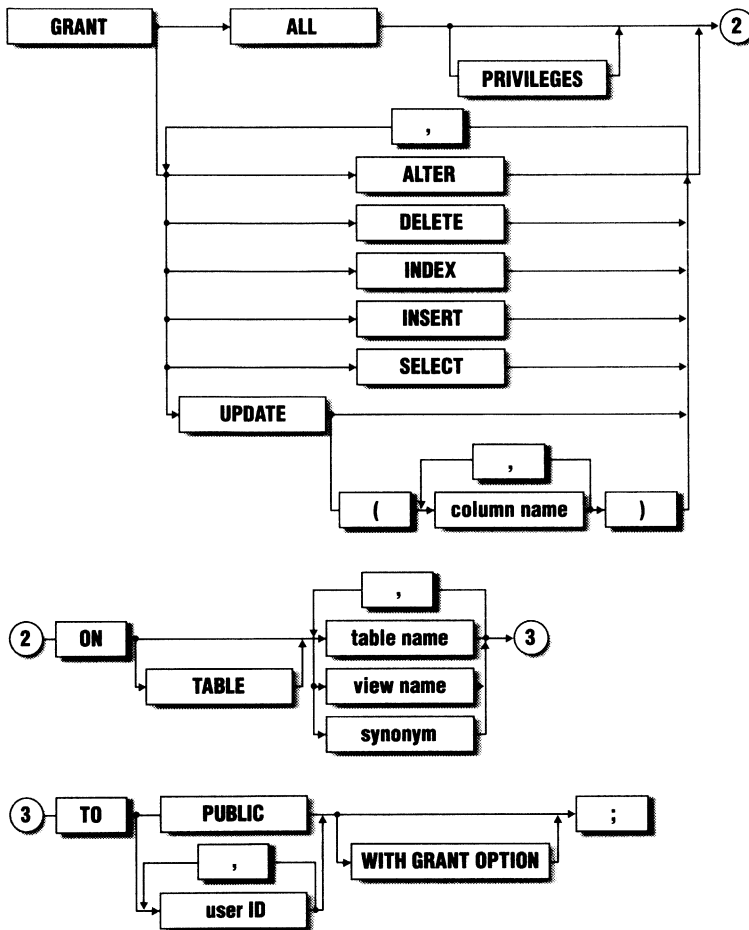


Figure 6-20 GRANT command

### 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 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**

PROTECT, 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>;
```

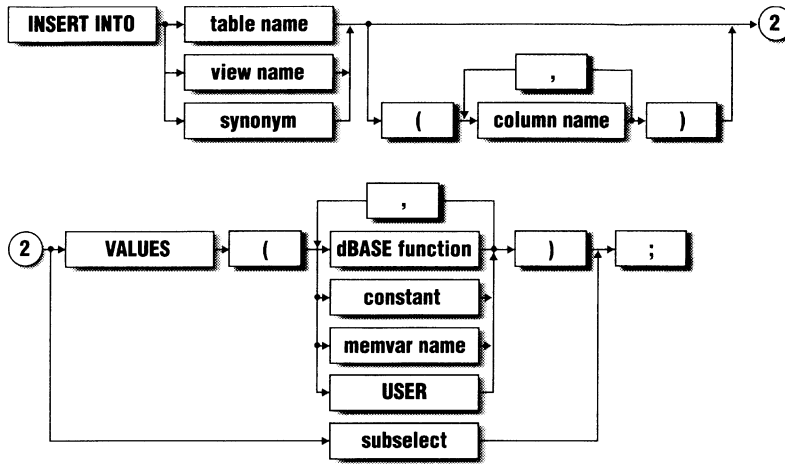


Figure 6-21 INSERT command

## 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 if PROTECT is installed).
- Are separated by commas. Character constants are enclosed by single or double quotes, or square brackets 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", 107/15/871,
"LOS ANGELES", "000001", 5678, 3.5);INSERT
```

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

## LOAD DATA

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>]];
```

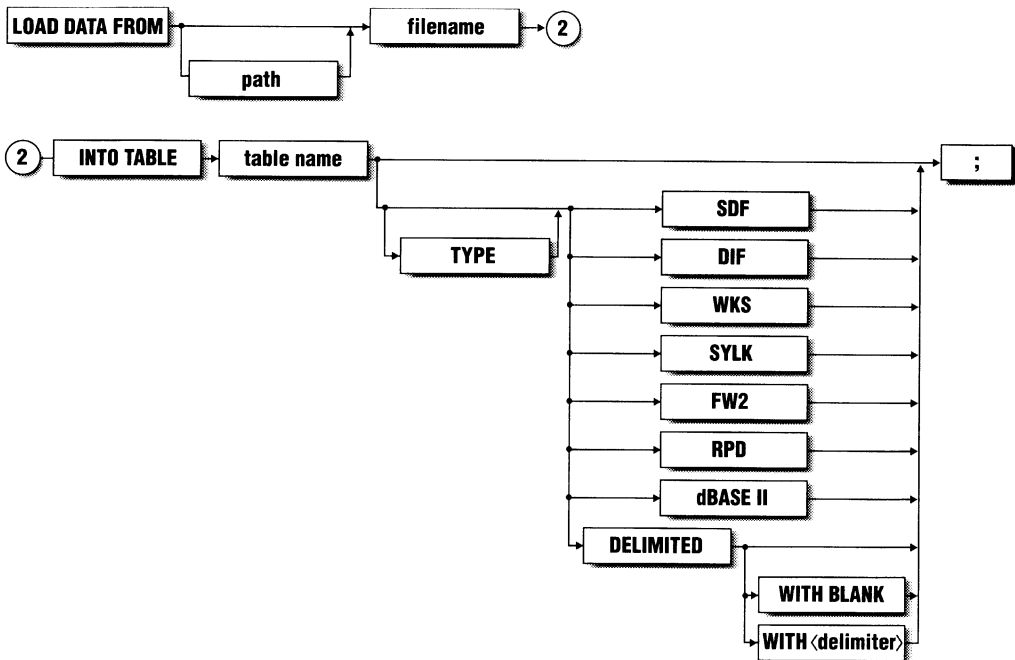


Figure 6-22 LOAD DATA command

## Usage

The LOAD DATA command supports the same file formats as the dBASE APPEND FROM command. You can use the LOAD DATA command to append 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, FW3, FW4)
- 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 directory or in another directory specified by an explicit 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 append 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 append 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 append data from a database file that has been encrypted by SQL or dBASE IV; however, you can append data from an unencrypted SQL table.*

## Example

To append data from a RapidFile file to the Staff table, type:

```
SQL. LOAD DATA FROM /x/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>;
```

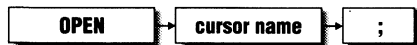


Figure 6-23 OPEN command

### 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>;
```

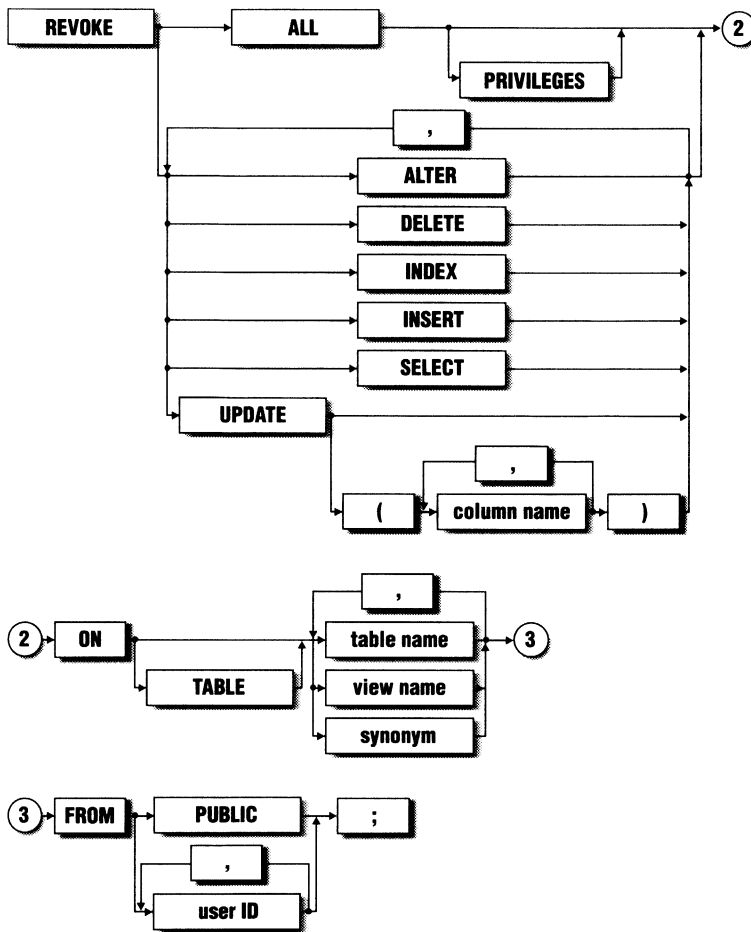


Figure 6-24 REVOKE command

**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

## REVOKE

- 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, PROTECT

## ROLLBACK

This command restores table data altered during an aborted transaction.

### Syntax

```
ROLLBACK [WORK];
```

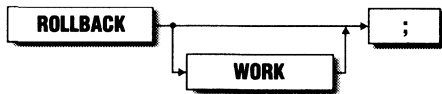


Figure 6-25 ROLLBACK command

### 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)
- Start-up 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
```

---

## RUNSTATS

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>];

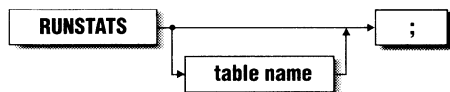


Figure 6-26 RUNSTATS command

### 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;
```

---

## SELECT

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 based on the values of columns specified in the SELECT clause. 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.

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.*

#### 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>...]
```

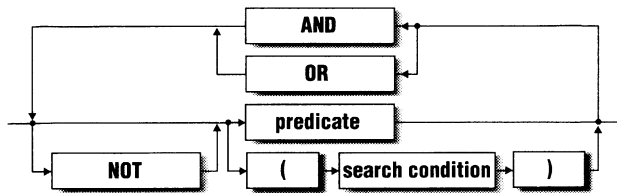


Figure 6-27 WHERE clause

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.

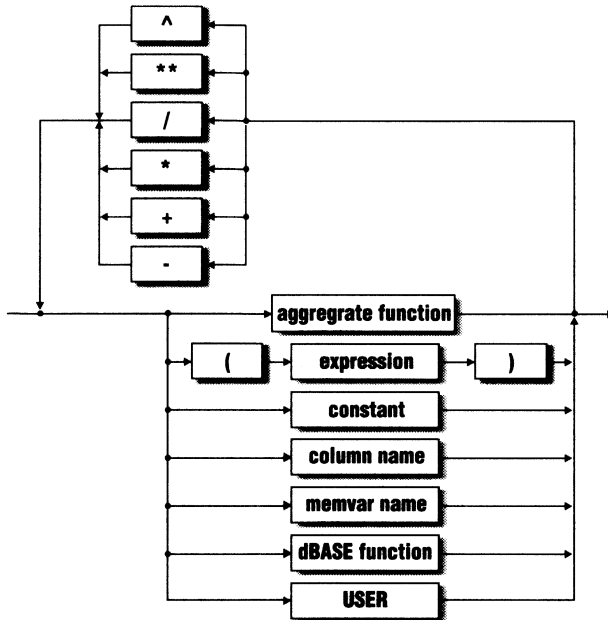


Figure 6-28 SQL expressions

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 uppercase letters are lower than the values of lowercase letters, the letter Z, for example, is evaluated as less than the value of letter z, 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.

### GROUP BY Clause

GROUP BY is an optional clause that lets you summarize information for a group 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.

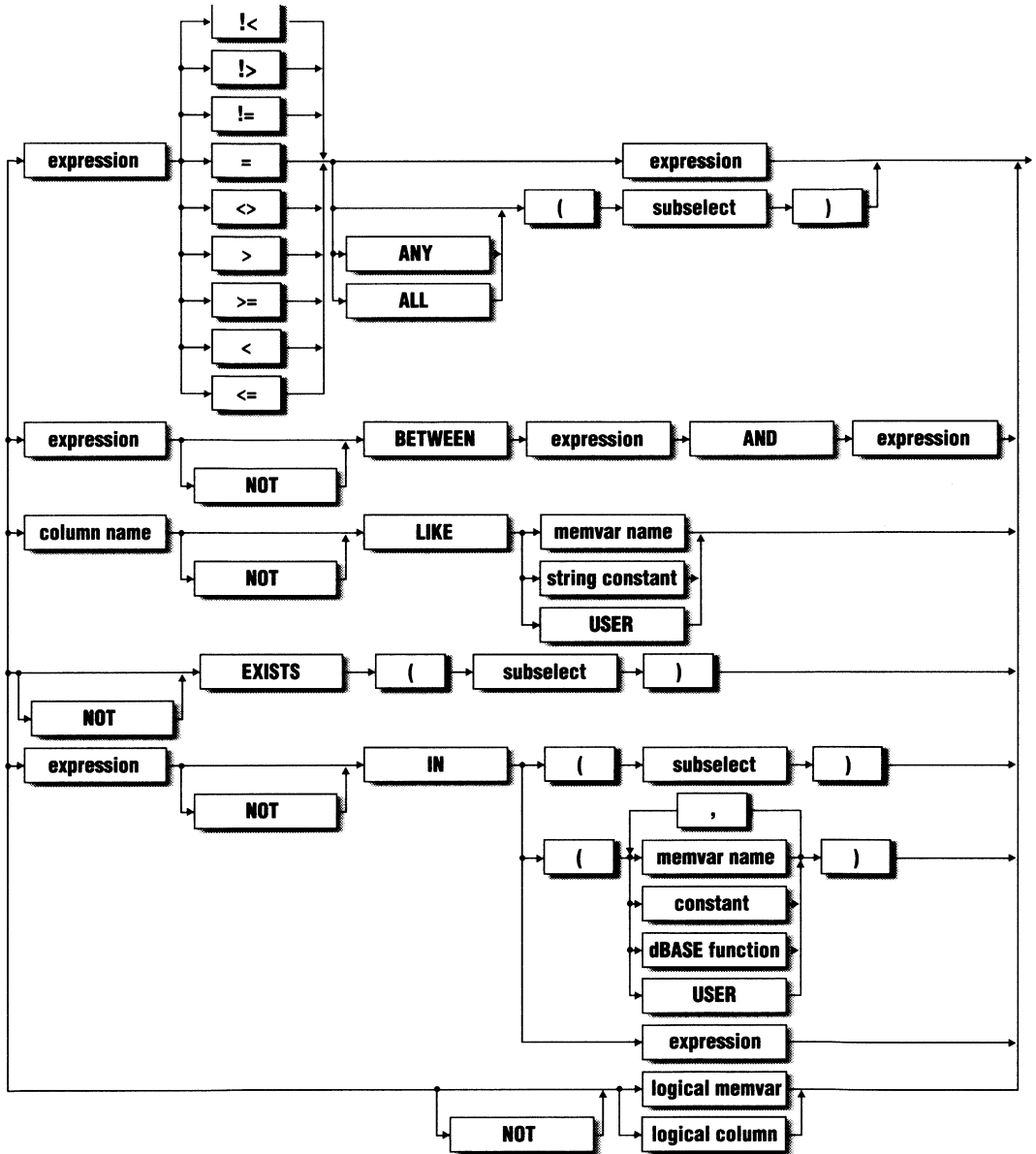


Figure 6-29 SQL predicates

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.

In interactive SQL, using the `KEEP` option saves the result table as a dBASE IV SQL table in the SQL database directory. In embedded SQL mode, the `KEEP` option initially saves the result table as a SQL table to the SQL database directory, then when the program (.prs) is exited, the SQL table is copied to the current directory as a permanent, unencrypted dBASE file. 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` command.



**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.

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.

# SELECT

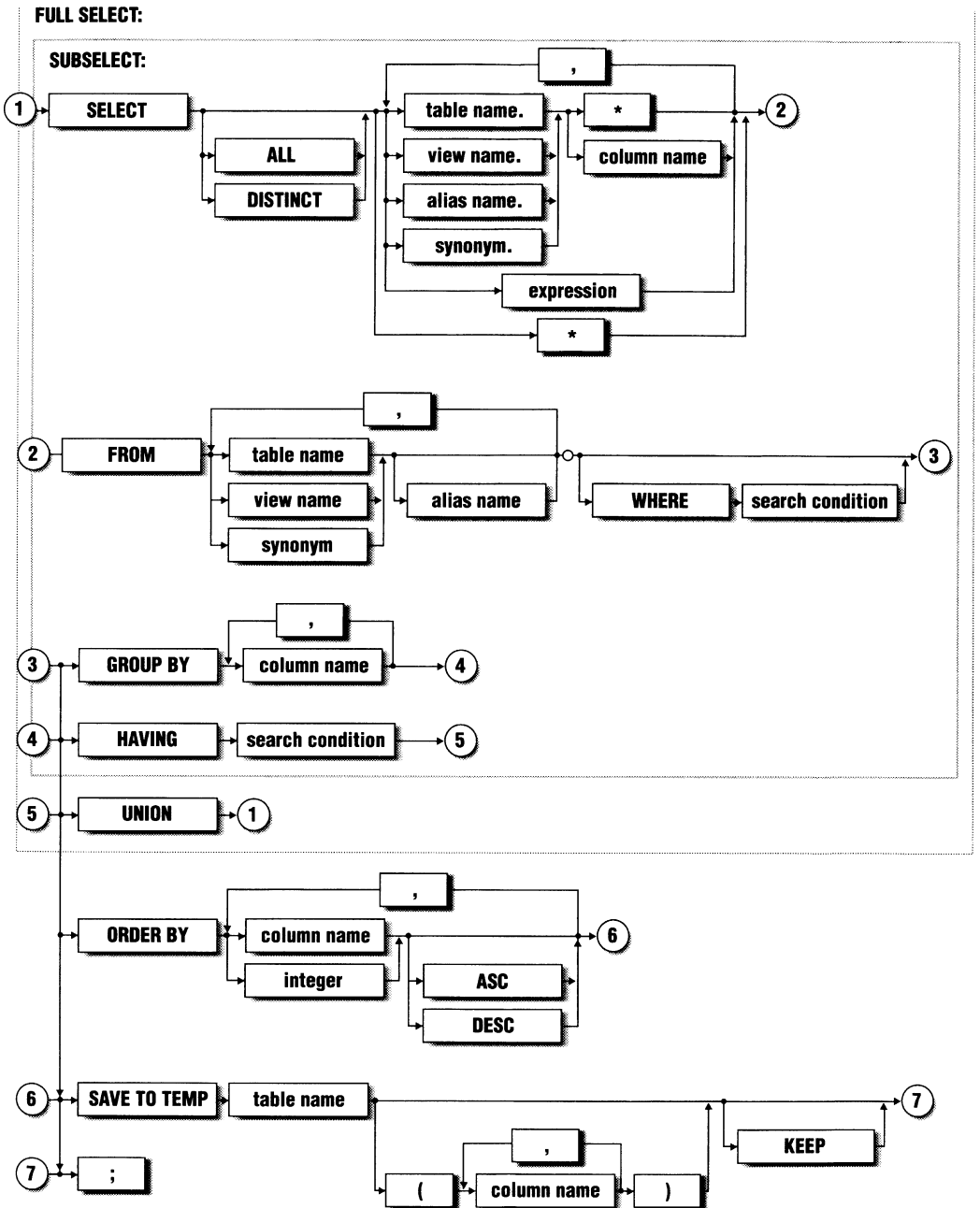


Figure 6-30 Display-only SELECT

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>...]];
```



**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.

# SELECT

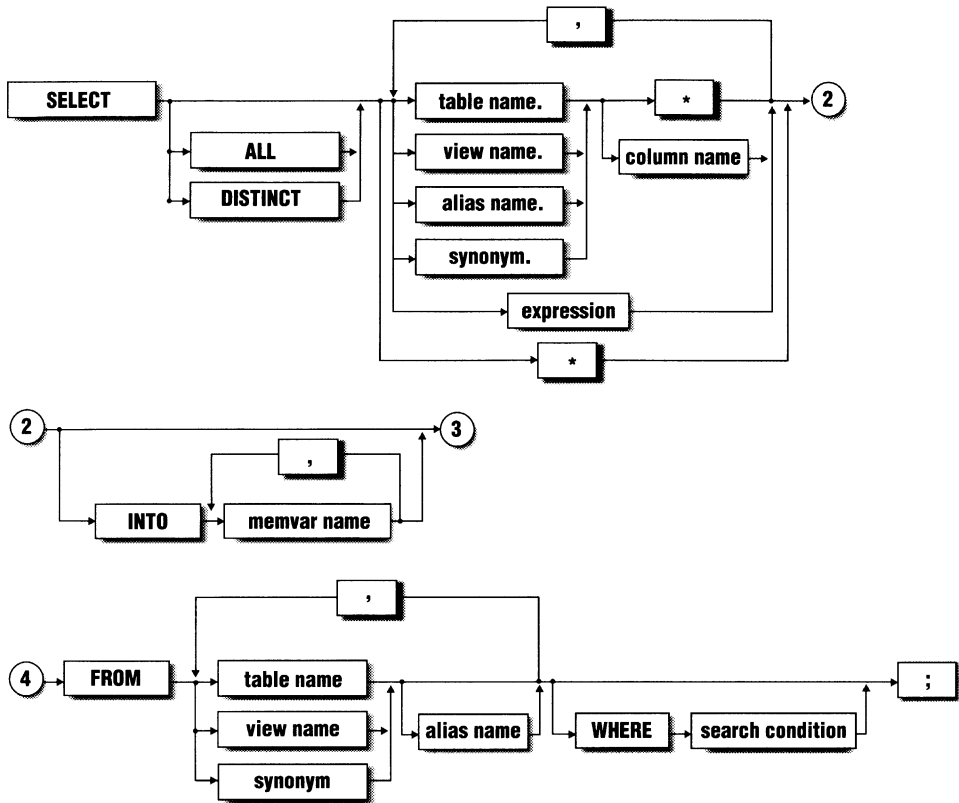


Figure 6-31 Embedded single-row SELECT (with INTO)

## 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 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

### 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 Inventory table, type:

```
SQL. SELECT DISTINCT Part_no, Descript, Unitcost  
    FROM Inventory;
```

To display information only for selected rows in the Inventory table, type:

```
SQL. SELECT Part_no, Descript, Location, On_hand  
    FROM Inventory  
    WHERE Location = "CHICAGO";
```

To display Inventory row values that satisfy several search conditions (grouped by parentheses), type:

```
SQL. SELECT Part_no, Descript, Location, On_hand, Unitcost  
    FROM Inventory  
    WHERE (Location = "LOS ANGELES" OR Location = "NEW YORK")  
    AND (On_hand < 20 AND Unitcost < 1000);
```

# SELECT

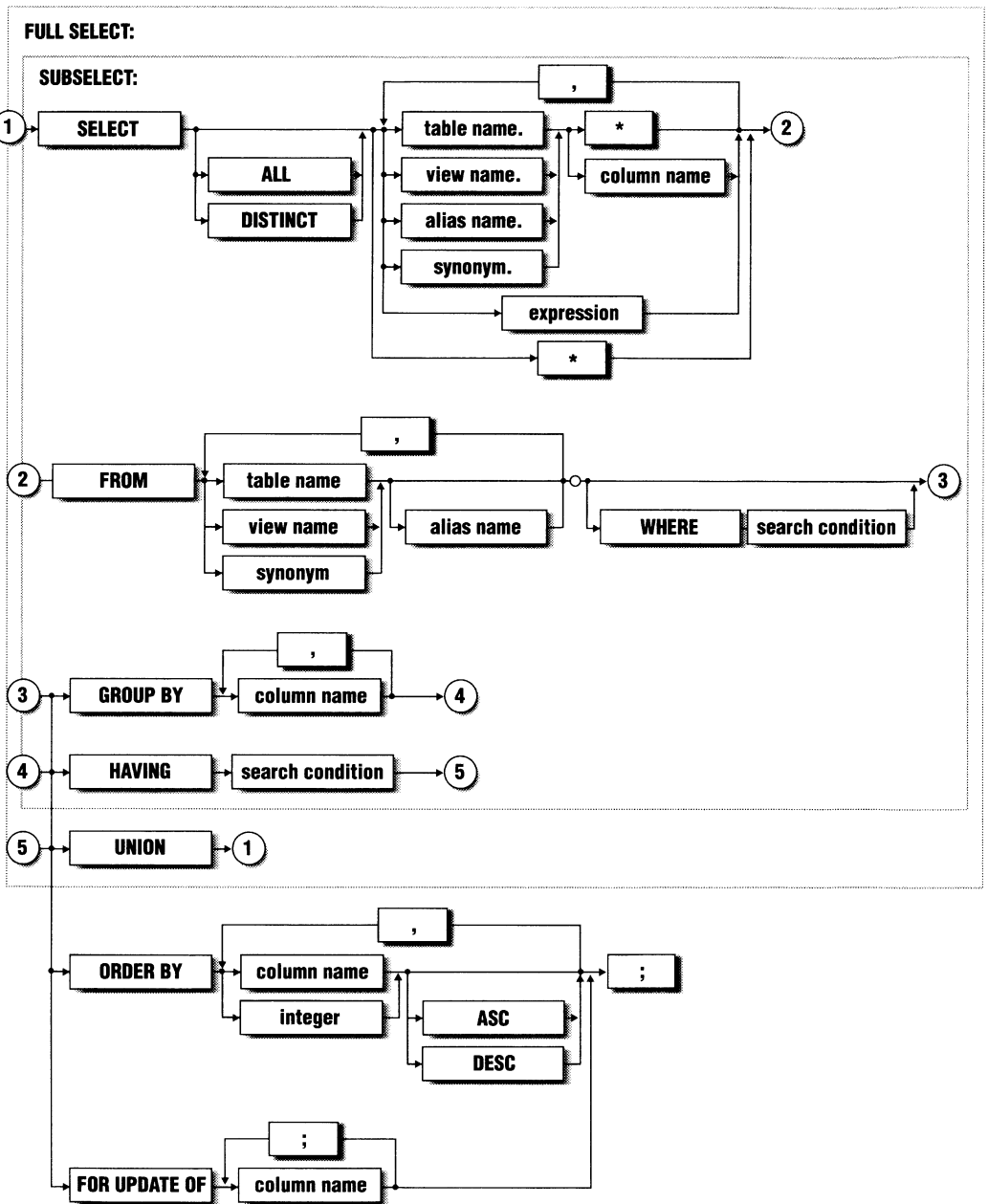


Figure 6-32 Embedded SELECT (with DECLARE CURSOR)

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 Inventory result rows using an ORDER BY clause, type:

```
SQL. SELECT Part_no, Descript, Location, On_hand
        FROM Inventory
        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 Inventory result rows by part number using GROUP BY, type:

```
SQL. SELECT Part_no, SUM(On_hand)
        FROM Inventory
        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
- Path of the database directory

This information is stored in the master catalog table, Sysdbs, in your SQL *home* directory.

 **NOTE** The *SQLHOME=* setting in *Config.db* specifies the path for your SQL *home* directory.

**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>];
```

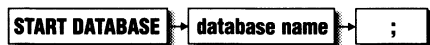


Figure 6-34 START DATABASE command

### 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 STOP 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>]];
```

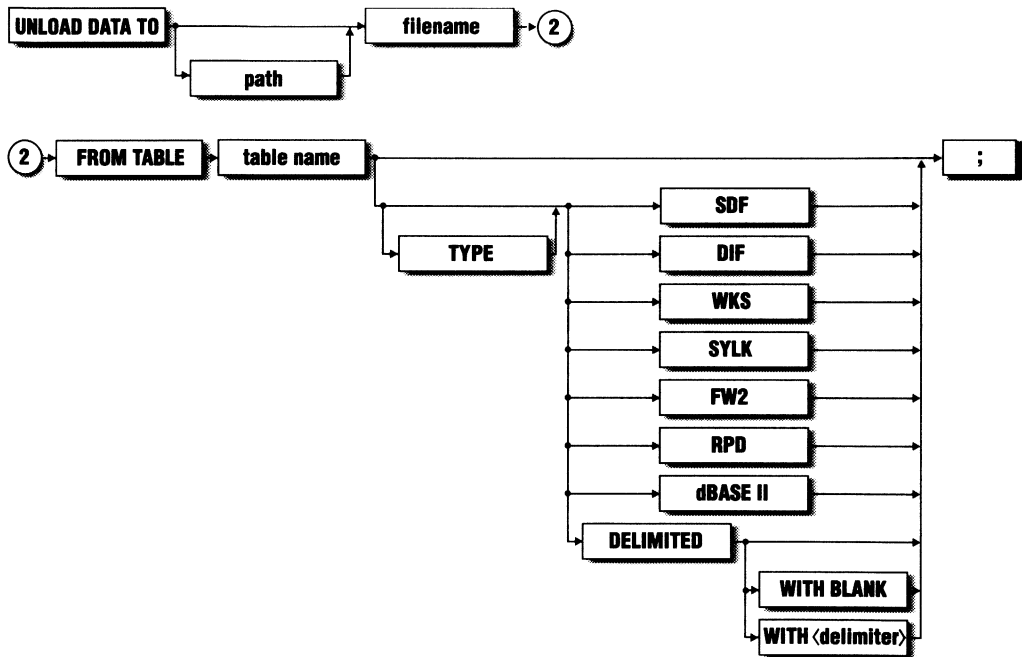


Figure 6-36 UNLOAD DATA command

### 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.

**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

## UPDATE

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>];
```

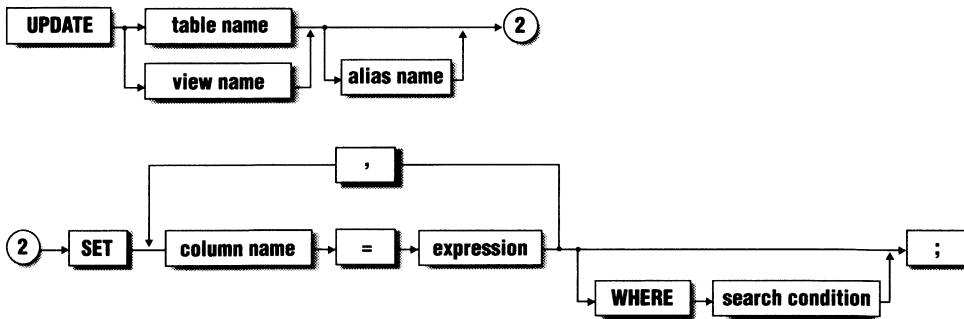


Figure 6-37 UPDATE command

## UPDATE

```
UPDATE <table name> (2)
  SET <column name> = <expression>
  [, <column name> = <expression>...]
  WHERE CURRENT OF <cursor name>;
```

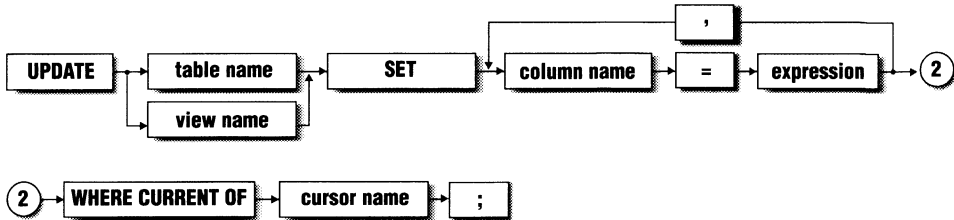


Figure 6-38 UPDATE command with WHERE CURRENT OF Clause

### 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.

**TIP** 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.

**NOTE** 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

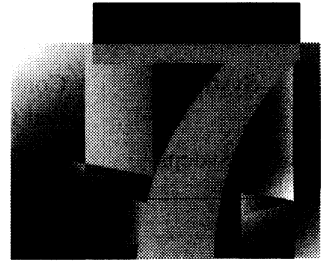




# SQL Catalogs



# 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 your SQL *home* directory 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. The full path is recorded in this table. If you change directories, dBASE IV will have trouble locating databases. 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.

---

<b>Table Name</b>	<b>Description</b>
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

---

(continued)

Table Name	Description
Systabls	Contains information describing every table and view — one row for every table or view defined in the current database
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:

---

<b>Column</b>	<b>Table</b>
COLCARD	Syscols
HIGH2KEY	Syscols
LOW2KEY	Syscols
FIRSTKCARD	Sysdxx
FULLKCARD	Sysidxx
NLEAF	Sysidxx
NLEVEL	Sysidxx
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.
COLLEN	Numeric(3)	Length of column.

---

*(continued)*

Column	Type	Description
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 your SQL *home* directory. One row of 91 bytes is added for each new SQL database created.

Column	Type	Description
TBNAME	Char(10)	Name of table or view
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)	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.

---

<b>Column</b>	<b>Type</b>	<b>Description</b>
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.

---

<b>Column</b>	<b>Type</b>	<b>Description</b>
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.

---

## Systemes Table

The Systemes 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)*

---

<b>Column</b>	<b>Type</b>	<b>Description</b>
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.

---

# Appendixes

dBASE IV Specifications

Sample Files

File Extensions

Structure of a Database (.dbf) file

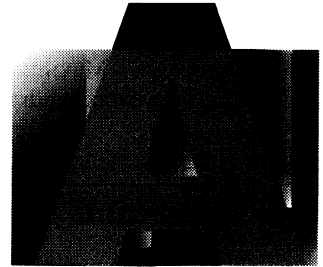
ASCII Chart

dBASE Commands and Functions Allowed in SQL Mode

dBASE Error Messages



# **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 (defaults: .ndx, 512 bytes; .mdx, 1,024 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 with dBASE internal editor. No limit on memo fields.

---

## **Arrays**

Dimensions: 2  
Total elements: As many as your system's memory will support; a single dimension, however, is limited to 65,535 elements.

---

## Multi-User Procedures

Locks — Maximum number of locks: 200 (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: 40  
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  
Open library files per run: 1  
Open SYSPROC 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:  $0.9 \times E+308$   
Smallest number:  $0.1 \times E-307$

Type N numbers:

10 to 20 digits based on the setting of SET PRECISION TO <expN>  
Largest number:  $0.9 \times E+308$   
Smallest number:  $0.1 \times E-307$

---

## 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

---



---

## 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 per macro library file

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: 40

Number of programmable function keys: 29

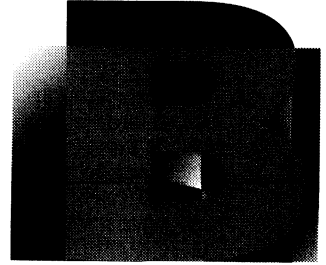
Number of recursive calls: 32

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 included with dBASE IV.

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-111-8050	SOFA, 8-FOOT	VELVET, BLUE, FRENCH	1200.00	1
4	87-107	C-222-1000	CHAIR, DESK	LEATHER, BROWN, HIGHBACK	1250.00	1
5	87-108	C-222-1000	CHAIR, DESK	LEATHER, BROWN, HIGHBACK	1250.00	1
6	87-109	C-400-2060	TABLE, END	WOOD, OAK, 2-FOOT, SQUARE	250.00	1
7	87-109	C-500-6050	LAMP, FLOOR	BRASS, 6-FOOT, ENGLISH	165.00	1
8	87-110	C-700-2020	FILE CABINET,2 DRAWER	METAL, BROWN	75.00	1
9	87-110	C-700-4020	FILE CABINET,4 DRAWER	METAL, BROWN	100.00	1
10	87-111	C-222-1001	CHAIR, DESK	LEATHER, BROWN	1000.00	1
11	87-112	C-222-3020	CHAIR, SIDE	PLASTIC, GREY	350.00	2
12	87-113	C-300-2020	BOOK CASE	WOOD, TEAK, 2-SHELF	125.00	1
13	87-114	C-500-6000	LAMP, FLOOR	BRASS, 6-FOOT, ART DECO	150.00	3
14	87-115	C-500-6050	LAMP, FLOOR	BRASS, 6-FOOT, ENGLISH	165.00	1
15	87-116	C-600-5000	DESK,EXECUTIVE 5-FOOT	WOOD, OAK, FANCY	1500.00	1
16	87-116	C-700-2030	FILE CABINET,2 DRAWER	METAL, BLACK	75.00	1
17	87-116	C-222-1001	CHAIR, DESK	LEATHER, BROWN	1000.00	1

The following is a list of the index keys used for the Stock.dbf file:

Production MDX file: STOCK.MDX

Index TAG: ORDER\_ID Key: ORDER\_ID

Index TAG: PART\_NAME Key: PART\_NAME

---

## Menus.prg

The following is a listing of Menus.prg, which produces a horizontal bar menu and four pop-up menus:

```
PRIVATE ll_status, ll_talk
IF SET( "TALK" ) = "ON"           && Save the talk status
  SET TALK OFF                   && Force talk off
  ll_talk = .T.
ELSE
  ll_talk = .F.
ENDIF
ll_status = SET( "STATUS" ) = "ON" && Save the state of status
SET STATUS OFF                   && Force status bar off

*-- Make sure a DBF or view is in use
IF ISBLANK( ALIAS( ) )
  USE ?                           && Bring up pick list of files
  IF ISBLANK( ALIAS( ) )          && No file in use yet
    DO Rest_Env
    RETURN                       && Return to calling program
  ENDIF
ENDIF

*-- Main Menu
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

  *-- Display these menus as you toggle between menu 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

  *-- Display this menu when you select the Exit pad
  ON SELECTION PAD Exit OF Main ACTIVATE POPUP Exit_pop

*-- Popup View_pop
DEFINE POPUP View_pop FROM 3,6 TO 8,21
  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( CHR(196), 16 ) SKIP
  DEFINE BAR 4 OF View_pop PROMPT "Delete"
  DO DelBarUp                       && Adjust bar for deleted record
  ON SELECTION POPUP View_pop DO View_pro

*-- Popup Goto_pop - No actions assigned
DEFINE POPUP Goto_pop FROM 3,18 TO 6,30
  DEFINE BAR 1 OF Goto_pop PROMPT "Skip"
  DEFINE BAR 2 OF Goto_pop PROMPT "Jump to"
```



```

*- Popup Prin_pop - No actions assigned
DEFINE POPUP Prin_pop FROM 3,32 TO 7,44
  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 3,40 TO 6,59
  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_pro

ACTIVATE MENU Main PAD View
RELEASE MENU Main
RELEASE POPUPS Exit_pop, Prin_pop, Goto_pop, View_pop
DO Rest_Env
RETURN

PROCEDURE Rest_Env
IF ll_status                                && If status bar was on
  SET STATUS ON                             && Turn it back on
ENDIF
IF ll_talk                                  && If talk was on
  SET TALK ON                               && Turn it back on
ENDIF
RETURN

PROCEDURE Exit_pro
DO CASE
  CASE BAR() = 1
    QUIT
  CASE BAR() = 2
    DEACTIVATE MENU
ENDCASE
RETURN

```

```

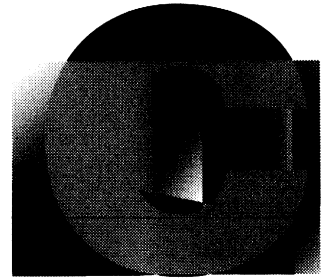
PROCEDURE View_pro
SAVE SCREEN TO temp
DO CASE
CASE BAR() = 1
  *- Allow the user to cancel from the append operation
  BEGIN TRANSACTION
  APPEND BLANK
  EDIT NEXT 1
  IF LASTKEY() = 27
    ROLLBACK
  ENDIF
  END TRANSACTION
  DO DelBarUp
CASE BAR() = 2
  EDIT NEXT 1
  DO DelBarUp
CASE BAR() = 4
  IF DELETED()
    RECALL
  ELSE
    DELETE
  ENDIF
  DO DelBarUp
ENDCASE
RESTORE SCREEN FROM temp
RELEASE SCREEN temp
RETURN

PROCEDURE DelBarUp
*- Update the Delete bar based on the record's delete flag status
IF DELETED()
  DEFINE BAR 4 OF View_pop PROMPT "Recall"
ELSE
  DEFINE BAR 4 OF View_pop PROMPT "Delete"
ENDIF
RETURN

*- END: 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 C-1 lists the extensions used and provides a brief description of the file content.

Table C-1 dBASE IV file extensions

---

<b>Extension</b>	<b>File Content</b>
.\$\$\$	Temporary file; action was not complete
.\$ab	Temporary file; virtual memory
.acc	Multi-user access control file
.app	Application design object file; Applications Generator only
.bak	Command, procedure, or database backup file
.bar	Horizontal bar design object file; Applications Generator only
.bch	Batch process design object file; Applications Generator only
.bin	Binary file
.cat	Catalog file
.cod	Template source file
.cpt	Encrypted memo file; used with password information (.crp) file
.crp	Password information file; created with PROTECT only
.cvt	A database file with change detection field
.db2	Renamed old dBASE II file; used for import and export

---

*(continued)*

Table C-1 dBASE IV file extensions (*continued*)

<b>Extension</b>	<b>File Content</b>
.log	Transaction log file
.db	Configuration file; for defaults on dBASE IV start-up
.dbf	Database file
.dbk	Database backup 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 TO
.doc	Documentation file; Applications Generator only
.err	Created if an error occurs during form generation, or if an unrecoverable error occurs
.fil	Files list design object file
.fmo	Compiled format (.fmt) file
.fmt	Generated format file; from .scr file
.fnl	Report binary name list 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, .fw3, .fw4	Framework spreadsheet or database file; used for import and export
.gen	Template file
.grp	Windows group file for dBASE IV
.hlp	dBASE IV help files
.ico	dBASE IV icon file under Windows
.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
.lnl	Label binary name list file

*(continued)*

Table C-1 dBASE IV file extensions (*continued*)

---

<b>Extension</b>	<b>File Content</b>
.mbk	Multiple index backup file
.mdx	Multiple index file
.mem	Memory file
.ndx	Single index file
.ovl	dBASE IV overlay file
.pif	Microsoft Windows file for non-Windows applications
.pop	Pop-up menu design object file; Applications Generator only
.pr2	Printer driver file
.prd	A file containing printer driver information for DBSETUP
.prf	Print form file
.prg	dBASE command or procedure file
.prs	dBASE SQL command or procedure file
.prt	Printer output file
.qbe	QBE query file
.qbo	Compiled QBE query (.qbe) file
.qry	dBASE III query file
.res	Resource file
.rpd	RapidFile file; used for import and export
.sc3	Renamed old dBASE III screen (.scr) file
.scr	Screen file
.snl	Screen binary name list file
.str	Structure list design object file; Applications Generator only
.t44/.w44	Intermediate work files; used by SORT and INDEX
.tbk	Database memo backup file
.txt	ASCII text output file
.upd	QBE update query file
.upo	Compiled QBE update query (.upd) file
.val	Values list design object file; Applications Generator only
.vmc	Configuration file; for Virtual Memory Manager (VMM)
.vue	View file
.win	Logical window save file
.wks, .wk1	Lotus 1-2-3 file; used with APPEND FROM and COPY TO

---

## File Relations and Compatibilities

Table C-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 C-2 File relations and compatibilities

File Type	dBase IV Extension	Compiled Object File	Backward Compatibility	Rename from dBASE III PLUS
Catalog	.cat		yes	
Database file	.dbf		yes *	
Memo	.dbt		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	
Create view from environment	.vue		yes	
dBASE language program	.prg	.dbo	no****	
SQL language program	.prs	.dbo	no	
Templates	.gen		no	
Macros	.key		no	
Index files	.ndx		yes	
Multiple index files	.mdx		no	

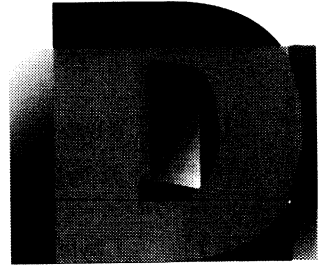
\*dBASE IV database files that contain type F numbers are not compatible with dBASE III PLUS. Use COPY TO <filename> TYPE DBMEMO3 to convert back to dBASE III PLUS. Also, numeric accuracy may be lost if a .dbf file is used in dBASE III PLUS, because dBASE IV allows numeric fields to contain up to 20 digits. (dBASE III PLUS 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 and convert type F numeric fields to type N numeric fields.

\*\*\* 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).

\*\*\*\*dBASE III PLUS may use your dBASE IV program file if it contains only dBASE III PLUS commands.

# 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 D-1 and D-2, provides information dBASE IV uses to maintain the database file.

Table D-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; fill with 0
14	1 byte	Flag indicating incomplete transaction*
15	1 byte	Encryption flag**
16-27	12 bytes	Reserved for dBASE IV in a multi-user environment

---

*(continued)*

Table D-1 Database file header (continued)

Byte	Contents	Meaning
28	1 byte	Production .mdx file flag; 01H if there is a production .mdx file, 00H if not
29–31	3 bytes	Reserved; fill with 0
32–n***	32 bytes each	Field descriptor array (the structure of this array is shown in Table D-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 D-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 D-3.



Table D-3 Allowable input for each data type

<b>Data Type</b>	<b>What it Accepts</b>
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.

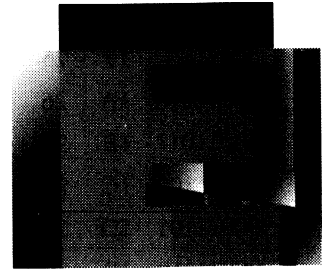
Each memo field of each record in the .dbf file contains the number of the block (in ASCII) where the memo field begins. If the memo field contains no data, the .dbf file contains blanks (20H) rather than a number.

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.



# ASCII Chart



Binary	Hex	Decimal	Character	Code	Symbol	Description
00000000	00	0	<null>	^@	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

(continued)

Binary	Hex	Decimal	Character	Code	Symbol	Description
00011000	18	24	↑	^X	CAN	Cancel
00011001	19	25	↓	^Y	EM	End of Medium
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

Binary	Hex	Decimal	Character	Binary	Hex	Decimal	Character
100000	20	32	<space>	110110	36	54	6
100001	21	33	!	110111	37	55	7
100010	22	34	"	111000	38	56	8
100011	23	35	#	111001	39	57	9
100100	24	36	\$	111010	3A	58	:
100101	25	37	%	111011	3B	59	;
100110	26	38	&	111100	3C	60	<
100111	27	39	'	111101	3D	61	=
101000	28	40	(	111110	3E	62	>
101001	29	41	)	111111	3F	63	?
101010	2A	42	*	1000000	40	64	@
101011	2B	43	+	1000001	41	65	A
101100	2C	44	,	1000010	42	66	B
101101	2D	45	-	1000011	43	67	C
101110	2E	46	.	1000100	44	68	D
101111	2F	47	/	1000101	45	69	E
110000	30	48	0	1000110	46	70	F
110001	31	49	1	1000111	47	71	G
110010	32	50	2	1001000	48	72	H
110011	33	51	3	1001001	49	73	I
110100	34	52	4	1001010	4A	74	J
110101	35	53	5	1001011	4B	75	K

(continued)

Binary	Hex	Decimal	Character	Binary	Hex	Decimal	Character
1001100	4C	76	L	1101101	6D	109	m
1001101	4D	77	M	1101110	6E	110	n
1001110	4E	78	N	1101111	6F	111	o
1001111	4F	79	O	1110000	70	112	p
1010000	50	80	P	1110001	71	113	q
1010001	51	81	Q	1110010	72	114	r
1010010	52	82	R	1110011	73	115	s
1010011	53	83	S	1110100	74	116	t
1010100	54	84	T	1110101	75	117	u
1010101	55	85	U	1110110	76	118	v
1010110	56	86	V	1110111	77	119	w
1010111	57	87	W	1111000	78	120	x
1011000	58	88	X	1111001	79	121	y
1011001	59	89	Y	1111010	7A	122	z
1011010	5A	90	Z	1111011	7B	123	{
1011011	5B	91	[	1111100	7C	124	
1011100	5C	92	\	1111101	7D	125	}
1011101	5D	93	]	1111110	7E	126	~
1011110	5E	94	^	1111111	7F	127	Δ
1011111	5F	95	_	10000000	80	128	Ç
1100000	60	96	`	10000001	81	129	ü
1100001	61	97	a	10000010	82	130	é
1100010	62	98	b	10000011	83	131	â
1100011	63	99	c	10000100	84	132	ä
1100100	64	100	d	10000101	85	133	à
1100101	65	101	e	10000110	86	134	â
1100110	66	102	f	10000111	87	135	ç
1100111	67	103	g	10001000	88	136	ê
1101000	68	104	h	10001001	89	137	ë
1101001	69	105	i	10001010	8A	138	è
1101010	6A	106	j	10001011	8B	139	ï
1101011	6B	107	k	10001100	8C	140	î
1101100	6C	108	l	10001101	8D	141	ì

(continued)

Binary	Hex	Decimal	Character	Binary	Hex	Decimal	Character
10001110	8E	142	Ä	10101111	AF	175	»
10001111	8F	143	Å	10110000	B0	176	⌘
10010000	90	144	É	10110001	B1	177	⌘
10010001	91	145	æ	10110010	B2	178	⌘
10010010	92	146	Æ	10110011	B3	179	⌘
10010011	93	147	ô	10110100	B4	180	⌘
10010100	94	148	ö	10110101	B5	181	⌘
10010101	95	149	ò	10110110	B6	182	⌘
10010110	96	150	ó	10110111	B7	183	⌘
10010111	97	151	ù	10111000	B8	184	⌘
10011000	98	152	ÿ	10111001	B9	185	⌘
10011001	99	153	ö	10111010	BA	186	⌘
10011010	9A	154	ü	10111011	BB	187	⌘
10011011	9B	155	φ	10111100	BC	188	⌘
10011100	9C	156	£	10111101	BD	189	⌘
10011101	9D	157	¥	10111110	BE	190	⌘
10011110	9E	158	ƒ	10111111	BF	191	⌘
10011111	9F	159	ƒ	11000000	C0	192	⌘
10100000	A0	160	á	11000001	C1	193	⌘
10100001	A1	161	í	11000010	C2	194	⌘
10100010	A2	162	ó	11000011	C3	195	⌘
10100011	A3	163	ú	11000100	C4	196	⌘
10100100	A4	164	ñ	11000101	C5	197	⌘
10100101	A5	165	ñ	11000110	C6	198	⌘
10100110	A6	166	æ	11000111	C7	199	⌘
10100111	A7	167	ø	11001000	C8	200	⌘
10101000	A8	168	ì	11001001	C9	201	⌘
10101001	A9	169	⌈	11001010	CA	202	⌘
10101010	AA	170	⌋	11001011	CB	203	⌘
10101011	AB	171	⌘	11001100	CC	204	⌘
10101100	AC	172	¼	11001101	CD	205	⌘
10101101	AD	173	ï	11001110	CE	206	⌘
10101110	AE	174	«	11001111	CF	207	⌘

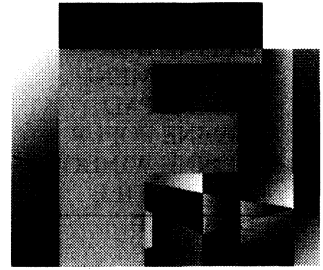
(continued)

Binary	Hex	Decimal	Character	Binary	Hex	Decimal	Character
11010000	D0	208	⏏	11101000	E8	232	⦶
11010001	D1	209	⏐	11101001	E9	233	⦷
11010010	D2	210	⏑	11101010	EA	234	⦸
11010011	D3	211	⏒	11101011	EB	235	⦹
11010100	D4	212	⏓	11101100	EC	236	⦺
11010101	D5	213	⏔	11101101	ED	237	⦻
11010110	D6	214	⏕	11101110	EE	238	⦼
11010111	D7	215	⏖	11101111	EF	239	⦽
11011000	D8	216	⏗	11110000	F0	240	≡
11011001	D9	217	⏘	11110001	F1	241	±
11011010	DA	218	⏙	11110010	F2	242	≥
11011011	DB	219	■	11110011	F3	243	≤
11011100	DC	220	■	11110100	F4	244	⌈
11011101	DD	221	■	11110101	F5	245	⌋
11011110	DE	222	■	11110110	F6	246	÷
11011111	DF	223	■	11110111	F7	247	≈
11100000	E0	224	α	11111000	F8	248	○
11100001	E1	225	β	11111001	F9	249	●
11100010	E2	226	Γ	11111010	FA	250	·
11100011	E3	227	Π	11111011	FB	251	√
11100100	E4	228	Σ	11111100	FC	252	⌈
11100101	E5	229	σ	11111101	FD	253	ε
11100110	E6	230	μ	11111110	FE	254	■
11100111	E7	231	τ	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 POPUPS
???	CLEAR TYPEAHEAD
@	CLEAR WINDOWS
@...CLEAR	CLOSE
@...FILL	CLOSE ALTERNATE
@...TO	CLOSE FORMAT
ACCEPT	CLOSE PROCEDURE
ACTIVATE MENU	COMPILE
ACTIVATE POPUP	COPY FILE
ACTIVATE SCREEN	CREATE
ACTIVATE WINDOW	CREATE FROM
APPEND	CREATE/MODIFY APPLICATION
ASSIST	CREATE/MODIFY LABEL
BEGIN/END TRANSACTION	CREATE/MODIFY QUERY/VIEW
BROWSE	CREATE/MODIFY REPORT
CALL	CREATE/MODIFY SCREEN
CANCEL	CREATE/MODIFY STRUCTURE
CHANGE	CREATE VIEW FROM
CLEAR	ENVIRONMENT
CLEAR GETS	DEACTIVATE MENU
CLEAR MEMORY	DEACTIVATE POPUP
CLEAR MENUS	DEACTIVATE WINDOW

DEBUG	PUBLIC
DECLARE	QUIT
DEFINE BAR	READ
DEFINE BOX	RECALL
DEFINE MENU	RELEASE
DEFINE PAD	RELEASE MENUS
DEFINE POPUP	RELEASE MODULE
DEFINE WINDOW	RELEASE POPUPS
DELETE FILE	RELEASE WINDOWS
DEXPORT	RENAME
DIR	REPORT FORM
DO	RESTORE
DO CASE	RESTORE MACROS
DO WHILE	RESTORE SCREEN
EDIT	RESTORE WINDOW
EJECT	RESUME
EJECT PAGE	RETRY
END TRANSACTION	RETURN
ERASE	RUN/!
EXIT	SAVE
FUNCTION	SAVE MACROS
HELP	SAVE SCREEN
IF	SAVE WINDOW
INPUT	SELECT
KEYBOARD	SET
LABEL FORM	SET ALTERNATE
LIST/DISPLAY FILES	SET AUTOSAVE
LIST/DISPLAY HISTORY	SET BELL
LIST/DISPLAY MEMORY	SET BORDER
LIST/DISPLAY STATUS	SET CATALOG
LIST/DISPLAY STRUCTURE	SET CENTURY
LIST/DISPLAY USERS	SET CLOCK
LOAD	SET COLOR
LOGOUT	SET CONFIRM
MODIFY COMMAND/FILE	SET CONSOLE
MOVE WINDOW	SET CURRENCY
NOTE/*/&&	SET CURSOR
ON ERROR/ESCAPE/KEY	SET DATE
ON PAD	SET DBTRAP
ON PAGE	SET DEBUG
ON READERROR	SET DECIMALS
ON SELECTION PAD	SET DEFAULT
ON SELECTION POPUP	SET DELETED
PACK	SET DELIMITERS
PARAMETERS	SET DESIGN
PLAY MACRO	SET DEVELOPMENT
PRINTJOB/ENDPRINTJOB	SET DEVICE
PRIVATE	SET DIRECTORY
PROCEDURE	SET DISPLAY
PROTECT	SET ECHO

SET ENCRYPTION	SET REFRESH
SET ESCAPE	SET REPROCESS
SET EXACT	SET SAFETY
SET EXCLUSIVE	SET SCOREBOARD
SET FORMAT	SET SEPARATOR
SET FULLPATH	SET SPACE
SET FUNCTION	SET SQL
SET HEADING	SET STATUS
SET HELP	SET STEP
SET HISTORY	SET TALK
SET HOURS	SET TITLE
SET INSTRUCT	SET TRAP
SET INTENSITY	SET TYPEAHEAD
SET LOCK	SET UNIQUE
SET MARGIN	SET VIEW
SET MARK	SET WINDOW
SET MESSAGE	SHOW MENU
SET NEAR	SHOW POPUP
SET ODOMETER	SORT
SET PATH	STORE
SET PAUSE	SUSPEND
SET POINT	TEXT
SET PRECISION	TYPE
SET PRINTER	USE
SET PROCEDURE	WAIT

---

## dBASE Functions Allowed in SQL Mode

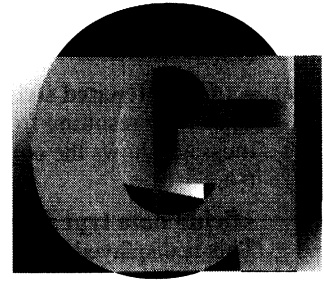


**NOTE** *Certain dBASE functions may be allowed in SQL mode but not within a SQL command, for example, in a SELECT or WHERE clause. Such functions are noted with an asterisk.*

&*	CHR()
ABS()	CMONTH()
ACOS()	COL()*
ALIAS()*	COMPLETED()*
ASC()	DATE()
ASIN()	DAY()
AT()	DESCENDING()
ATAN()	DIFFERENCE()
ATN2()	DISKSPACE()*
BAR()*	DMY()
CALL()*	DOW()
CDOW()	DTOC()
CEILING()	DTOR()
CERROR()*	DTOS()
COS()	ERROR()*
CTOD()	EXP()

FCLOSE()	MIN()*
FCREATE()	MOD()
FDATE()*	MONTH()
FEOF()	NETWORK()*
FERROR()	OS()*
FFLUSH()	PAD()*
FGETS()	PCOL()*
FILE()*	PI()
FIXED()	POPUP()*
FKLABEL()*	PRINTSTATUS()*
FKMAX()*	PROGRAM()*
FLOAT()	PROMPT()*
FLOOR()	PROW()*
FOPEN()	RAND()
FOR()	READKEY()*
FPUTS()	REPLICATE()
FREAD()	RIGHT()
FSEEK()	ROLLBACK()*
FSIZE()*	ROUND()
FTIME()*	ROW()*
FWRITE()	RTOD()
GETENV()*	RTRIM()
HOME()	SELECT()*
ID()	SET()*
IIF()*	SIGN()
INKEY()*	SIN()
INT()	SOUNDEX()
ISALPHA()*	SPACE()
ISCOLOR()*	SQRT()
ISLOWER()*	STR()
ISUPPER()*	STUFF()
KEY()*	SUBSTR()
LASTKEY()*	TAGCOUNT()
LEFT()	TAGNO()
LEN()	TAN()
LIKE()*	TIME()
LINENO()*	TRANSFORM()
LOG()	TRIM()
LOG10()	TYPE()*
LOWER()	UPPER()
LTRIM()	USER()
MAX()*	VAL()
MDY()	VARREAD()*
MEMORY()*	VERSION()*
MENU()*	WINDOW()*
MESSAGE()*	YEAR()

# dBASE Error Messages



---

## Unrecoverable Errors

When dBASE terminates abnormally with an unrecoverable error, dBASE attempts to identify the error. If it can identify the error, dBASE displays a message such as “Internal error ##### occurred while executing dBASE. This error is unrecoverable and dBASE is terminating.”

dBASE also automatically creates a file, Dbase.err, in the dBASE home directory and writes data to it when an unrecoverable error occurs. You can use the information in Dbase.err as a troubleshooting tool. It contains the following information:

- The dBASE version
- The date and time of the error
- Registers (if a protection violation occurred)
- The internal error code
- The status of memory allocation
- A list and status of open files
- The line of code dBASE was executing when the error occurred
- A trace of the C program stack

For descriptions of the internal error codes, refer to Errcode.txt, located in the dBASE home directory. This file lists and describes briefly all the internal error codes that dBASE detects and reports.

If you still need help with the problem after you read the Dbase.err file, contact Technical Support and have the error information ready.

---

## Error Messages

This section lists all the dBASE error messages in alphabetical order. Error messages with numbers can be trapped with the ON ERROR command. To look up an error message by number, use the on-line help.

**+ : Concatenated string too large** [77]  
The string resulting from the concatenation of two strings exceeds 254 characters. This message shows the concatenation operator used to create the large string was a plus (+).

**- : Concatenated string too large** [76]  
The string resulting from the concatenation of two strings exceeds 254 characters. This message shows the concatenation operator used to create the large string was a minus (-).

**<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.

**A REPLACE query must have at least one WITH operator** [370]  
You tried to run an update query (REPLACE) and there is no WITH operator.

**A tag has been marked NOSAVE in: <.mdx filename>** [520]  
You are trying to use a tag in a non-production .mdx file. That .mdx file contains a tag which another work area has marked with the NOSAVE option. You cannot use this .mdx file until the other work area closes the .mdx file or removes the NOSAVE option from the tag.

**A UNIQUE aggregate must be the only aggregate in a QUERY** [324]  
An aggregate QUERY can have only one UNIQUE in it.

**Access Control initialization error** [501]  
There is a problem handling your access control file. Please contact your system administrator.

**Access file corrupted** [500]  
The user access control file on the local area network is not usable. See your system administrator.

**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.

**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 expression not in range** [232]  
You are using an alias expression outside the range of 1 through 40.

**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.

**ALIAS name already in use** [24]  
You attempted to USE a database file that is already open, has the same alias as another open file, or has a name or alias within the range of A through J. dBASE IV reserves this range for default aliases.

**ALIAS not found: <alias name>** [13]  
You attempted to SELECT a database area outside the allowable ranges (A–J or 1–40), or you 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.

**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.

**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.

**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.

**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.

**Asterisk (\*) 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.

**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]

You either used an illegal value (such as a zero) during the declaration of an array, or you declared an array that has more than 65,535 elements in a dimension, or you declared an array that has more than two dimensions.

**Bad PROCEDURE name** [32]\*

You have used a PROCEDURE statement in a program. The name assigned to the procedure is invalid. Please enter a correct name.

**Badly formed subquery** [1171]

A predicate in an outer query has been placed in a nested query. Rewrite the query without a subquery (that is, use the outer predicate to limit the rows that are retrieved).

**BAR not DEFINED** [524]

You are attempting to use a pop-up menu bar that you have not yet defined.

**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 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.

**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 is needed.

**Calculated field requires an expression**

You tried to create a calculated field without an expression. You pressed **Ctrl-End** in the validation submenu, and no expression had been entered.



- Calculated fields in the VIEW must not be empty** [323]  
You deleted the expression of a calculated field that is in the view.
- 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.
- Cannot access SQL-only database** [405]  
You have tried to use an invalid dBASE IV command against a database that exists only on the server. Either use a different command or make a local copy of the data.
- 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 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 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]  
An active menu on the screen cannot be cleared with the CLEAR MENU or RELEASE MENU command. First use the DEACTIVATE MENU command, or activate a different menu.
- Cannot clear popup in use** [177]  
A pop-up menu active on the screen cannot be cleared with the CLEAR POPUPS or RELEASE POPUPS command.
- Cannot CLEAR WINDOW when menu is active** [241]  
You cannot CLEAR a window that is displaying an active menu.
- 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 backup file: <filename>** [212]  
The MODIFY STRUCTURE command was unable to make its backup files because the current disk is full. Make room on the disk before changing a file's structure.

- Cannot CREATE INDEX/GROUP BY/ORDER BY/SELECT DISTINCT/  
UNION on 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 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 DEACTIVATE WINDOW when menu is active** [239]  
You cannot deactivate the window which is displaying an active menu.
- Cannot DEACTIVATE WINDOW when popup is active** [234]  
You cannot deactivate the window which is displaying the active popup menu.
- Cannot delete SQL created TAG: <tagname>** [376]  
Production .mdx files may contain tags created in SQL mode and dBASE mode. This message indicates that you tried to delete an index tag in dBASE mode that was made during SQL mode.
- 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 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 erase open file** [89]  
Close database or index files before attempting to ERASE them.
- 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 or return to previous design screen if errors exist**[344]  
This message occurs when you attempt to go to Browse/Edit when there is an error in the query design.
- 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 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 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 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 mark NOSAVE, file in use: <.mdx filename>** [521]  
 You have requested a temporary tag with the NOSAVE option. The .mdx file containing the tag is already in use by another user and so cannot be marked NOSAVE.
- 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 modify SQL table** [375]  
 You cannot modify a SQL table in dBASE IV mode; you must be in interactive SQL mode.
- Cannot move an 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 of the layout**  
 You are trying to move the cursor below the allowed area of the layout surface.
- Cannot move the cursor beyond the left edge of the layout**  
 You are trying to move the cursor too far to the left on a design surface.
- Cannot move the cursor beyond the right edge of the layout**  
 You are trying to move the cursor too far to the right on a design surface.
- Cannot move the cursor beyond the top of the layout**  
 You are trying to move the cursor too high on a design surface.
- 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 may occur when trying to use the Applications Generator and dbase2.res is missing. You may try to locate this .res file on your original program media and copy it to the directory from which you start dBASE IV. If you cannot locate it, see your system administrator for assistance.
- Cannot overwrite a file when transaction is in process** [407]  
 You cannot use any command that overwrites a file that is involved in a transaction. This would make ROLLBACK not possible.

- Cannot overwrite a tag during a transaction** [516]  
An .mdx index file cannot be altered while a transaction is in progress. You must issue an END TRANSACTION command before you can change an index tag.
- Cannot overwrite active macro library** [408]  
You are using the RESTORE MACRO command while running a macro. You cannot overwrite the current macro library until the macro has finished executing.
- 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 place a link in a field that has an error** [346]  
You are trying to link to a field that has an error.
- Cannot play back the macro being recorded** [267]  
You must finish recording a keyboard macro before you can play it back.
- Cannot re-define menu in use** [174]  
You are attempting to DEFINE a MENU with the same name as the currently active menu. Use a different name for the new menu that is being DEFINEd. The new menu is probably being DEFINEd during an interruption triggered by ON SELECTION, ON KEY, or ON ERROR.
- Cannot re-define popup in use** [175]  
You are attempting to DEFINE a POPUP with the same name as the currently active popup. Use a different name for the new popup that is being DEFINEd. The new popup is probably being DEFINEd during an interruption triggered by ON SELECTION, ON KEY, or ON ERROR.
- 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.
- Cannot recompile <object filename>, source file missing** [144]  
The source file for the object file named in the message is missing. If you cannot locate the source code file, you must recreate the object from the Control Center design surfaces.
- 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 RELEASE WINDOW when menu is active** [240]  
You cannot RELEASE the window that is displaying an active menu.
- Cannot RELEASE WINDOW when popup is active** [237]  
You cannot release the window which is displaying the active pop-up menu.
- Cannot select requested database** [17]  
You have selected an invalid work area number. Use a number from 1 to 40 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. 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 there is no current transaction, the database may have been flagged by an earlier unsuccessful ROLLBACK. Clear the integrity flag from the database file with the RESET command.

**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]**

Before you can SET CATALOG ON, you must establish a catalog with SET CATALOG TO.

**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 DBSETUP setting, make sure the complete set of SQL system tables, including Sysdbs, has been transferred to the new SQLHOME directory.

**Catalog table(s) locked by another user : <catalog 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.

**Catalog tables are read-only : <table name> [1161]**

Catalog table updates are only allowed on a password-protected system by the SQLDBA superuser ID.

**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.

**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.

**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.

**CMDSET():** [66]

This error may occur if you use a .dbo file from an earlier version of dBASE IV. Recompile all previous version program and procedure files. This internal error message displays if a .dbo file contains token code that can be interpreted as an illegal SET command token code.

**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/SUM/COUNT 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 number must be between 0 and either right margin or 255** [223]

You have specified a printer column number that is too large. Columns wider than 255 are not allowed. In addition, if \_wrap is .T., the printer column cannot be larger than \_rmargin. You can specify the printer column with the ???...AT <expN> command or by assigning a value to \_pcolno.

**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.

**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.

**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 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 information about commands that cannot be used recursively.

**Command cannot be executed within a transaction [1169]**

SQL data definition commands ALTER, CREATE, DBCHECK, DBDEFINE, DROP, GRANT, REVOKE, and RUNSTATS are not allowed in transactions. Run these commands outside of transactions.

**Command not allowed in format files [130]**

Commands like COMPILE, DO, SET PROCEDURE are not allowed in format files, or inside UDFs used in format files.

**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. You can SET SQL OFF and try again, or else use an equivalent SQL command.

**Command not functional in dBASE IV [93]**

This is a warning message, and not an error. 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]**

You are using COMPILE...RUNTIME to create a RunTime program file. Certain commands like MODIFY STRUCTURE and CREATE SCREEN are not allowed in a RunTime program. You have a .prg 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.

**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.

**Compilation error** [360]

A program file would not compile because of syntax errors. Correct the errors in the program file and recompile.

**Compiler directive not supported** [492]

Some of the compiler directives common to other compilers are not supported by dBASE IV. You can only use:

- \* #define
- \* #undefine
- \* #ifdef
- \* #ifndef
- \* #else
- \* #endif

**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.

**CONTINUE without LOCATE** [42]

You cannot CONTINUE without executing a LOCATE command.

**Control file could not be found** [505]

You are trying to load dBASE IV in a network environment and the access control file (dbase412.acc) cannot be found. Make sure you have this file. You may need to specify a path to the control file by using the CONTROLPATH setting in your Config.db file.

**Coordinates are off the screen** [30]

You are using row and column numbers that are off the screen when defining or moving a window. With STATUS OFF, in 25-line mode, the screen limits are 0,0 to 24,79.

**Copying production .MDX requires one free work area** [399]

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 another printer driver. See the Dbdriver.txt file on the Install disk for a list of dBASE IV printer drivers and the attributes they support. If the driver you need is not on your hard disk, extract it from the Drivers.exe file.

Assign the printer driver file (.pr2) that supports quality printing to the \_pdriver system memory variable. From the dot prompt, type:

```
_pdriver='filename.pr2'
```

**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 .prs 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.

**Cyclic relation** [44]

You created an endless loop by setting a relation that eventually cycles back to the currently selected database file. Use SET RELATION TO with no parameters to disconnect the relation in the currently selected work area.

**Data Error**

The operating system has detected a bad sector on your hard disk. This is a critical hardware error that may cause loss of data.

**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** [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 KEYMATCH( ) that tries to match expressions of 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 an operator applied to a field or variable whose data type is inappropriate for the operator.

**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 encrypted** [131]

You are attempting to use an encrypted database without going through PROTECT and the log-in screen to enter your password.

**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.

**Database not indexed** [26]

If you attempted to execute a FIND, KEYMATCH( ), SEEK, or UPDATE command on a database file, you need to open an index file for the database file. If you attempted to SET a RELATION to a database file and received this error, the database file must first be indexed.

**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 = TR.

**DBDEFINE completed; error/warnings found** [2013]

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.

**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.

**Delete old MDX file?** [366]

You are trying to create a new .dbf file, which will overwrite an existing .mdx file. To avoid this, choose an alternate database filename.

**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 : <table 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.

**Disk full when writing file: <filename>** [56]

The disk you are trying to write to is full. Make room on the target disk by removing (deleting) files.

**Disk I/O error**

The internal editor experienced a problem writing or reading to the disk.

**Disk is write protected**

You are attempting to write to write-protected media.

**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.

**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.

**Drive not ready on <drive>**

The drive name is appended to this message. The drive you are attempting to read from or write to is not ready.

**Duplicate field name**

You entered a duplicate field name when creating or modifying a database structure.

**Duplicate field name: <field name>** [264]

You are using CREATE FROM and the FIELD\_NAME field in the source file duplicates a name already used by another record in the source file. Use another name.

**Duplicate sort number** [322]

During a semantic check, dBASE IV has found an identical sort directive in two different columns (query design).

**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.

**Editing condition not satisfied** [171]  
The VALID expression in an @...GET command was not satisfied. Enter the correct data or press ESCAPE to leave the surface. You can trap for this error with ON READERROR.

**Empty structure cannot be saved**  
You pressed **Ctrl-W** or **Ctrl-End** to save an empty database structure from the database design screen. You cannot save a .dbf file that has no fields in its structure.

**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.

**Enter RESUME to return to the Debugger** [401]  
You are trying to use the Debugger, but it is already in use. Use the RESUME command to return to the active debugging session.

You can then enter Q within the debugger to terminate that session.

**Entry already exists in catalog table Syscols for the following table, column:**  
**<table name> <column name>**

DBDEFINE has attempted to define a table which is already in the Syscols catalog table. This 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 name> <index name>**

DBDEFINE has attempted to define an index which 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 name><index name><column name>**

DBDEFINE has attempted to define an index key which 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.

- 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.
- Equal sign expected** [1022]  
The equal sign is missing or misplaced after the keyword SET <column name> in an UPDATE statement.
- Error after aggregate operator** [509]  
An aggregate operator such as MIN, MAX, SUM, AVG is followed by an illegal construct.
- Error after the EVERY operator** [510]  
The keyword EVERY must be followed by an example variable.
- 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 GROUP BY operator** [508]  
The keyword GROUP must be followed by the word BY. Check if missing or misspelled.
- Error in operation in filename column** [514]  
Only operations like MARK, UNMARK, and APPEND are allowed in the filename column.
- 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 a syntax error has occurred.
- Error with LIKE operator** [512]  
The LIKE operator must be followed by a quoted string.
- Error with SOUNDS LIKE operator** [511]  
The SOUNDS LIKE operator must be followed by a quoted string. Make sure you have written LIKE and that the string has opening and closing quotes.
- Errors in query definition - cannot be executed** [352]  
The query file you are trying to use has errors and cannot be executed. You either pressed **F2** from the Control Center on an invalid query file, or you SET VIEW TO an invalid query file.
- Exceeded internal register allocation, simplify command** [146]  
A command or nested procedure group exceeds the amount of space allotted for it internally by dBASE. Therefore, the command expression cannot be parsed. Simplify the command.

You can also get this message when trying to pass too many literal parameters to a procedure.

**Exceeded maximum compiler nesting level** [248]

dBASE IV allows a 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 nesting level for SCAN command** [163]

You have exceeded the maximum number of commands SCAN can process. Reduce the number of commands and process the database file in several SCAN passes if necessary.

**Exceeded maximum number of compile time symbol references** [273]

Each command is allowed only 256 references to be resolved at compile time. These references may be field names, memory variables, filenames, or other expressions. Simplify your command.

**Exceeded maximum number of compile time symbols** [274]

Compile time symbols are the unique names of memory variables and database fields in a compiled procedure. The default maximum number of allowable symbols is 500. You have exceeded this number. You can reduce the number of symbols in the procedure, or change the maximum number of allowable symbols in Config.db by changing the value assigned to CTMAXSYMS.

**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*.

**Exclusive use of database is required** [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.

**Execution error on** [80]

This is a prefix to other messages. It cannot be trapped. Look up the rest of the text in the message for an explanation. Examples:

Execution error on STR(): Out of range

Look up STR(): Out of range

**Expression not allowed in GROUP BY/ORDER BY clause** [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.

**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 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 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 have specified a field that cannot be accessed by the command. Use DISPLAY STRUCTURE to see if the field is in the current database file and is active. You may need to use SET FIELDS ON or SET FIELDS TO to make the field available to the command you are trying to use.

This message is returned by some commands, such as SORT, that use field names.

**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 requires a name**

This is a 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.

**Field type must be C, N, F, D, L, or M**

You pressed the wrong key while changing the field type on the database design screen.

**Field type must be Y or N**

This message 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]

You are using a command such as BLANK, SET FIELDS TO, or @ GET, and you have tried to apply the command to more than 255 fields. Specify fewer fields for this command.

**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 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 catalog empty**  
There are no files in the catalog you are attempting to use. Use another catalog or create a catalog and include the files you want in it.
- File does not exist: <filename>** [1]  
The specified file cannot be found. Verify the filename, and check the current path for the file's existence.
- File encryption error** [1282]  
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 been deleted** [49]  
When you delete a file with SET TALK ON, this message confirms the deletion.
- 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 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. You will also receive this message when using the SET INDEX...NOSAVE option if another user has opened the .mdx file you want to use.
- 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 : <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.



- File not accessible** [29]  
You are trying to write to or create a disk file that has an access restriction placed on it by the operating system.
- File not active in indicated workarea: <indexname>** [518]  
The index file named in the error message is not active in the work area you have chosen. Activate the index file in the chosen work area first.
- File not an MDX index file** [206]  
The file you attempted to open is not a valid .mdx file.
- 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 not in transaction log** [194]  
You tried to perform a ROLLBACK on a file that was not recorded in a transaction log file. You must restore the file to its pre-transaction state from a backup copy.
- 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 not open in current work area: <filename>** [179]  
An index file you specified for COPY TAG, DELETE TAG, or COPY INDEX is not open in the current work area.
- File open error : <filename>** [1280]  
There was a problem opening a file.
- File read error : <filename>** [1277]  
There was a problem reading a file.
- File seek error : <filename>** [1276]  
There was a problem with a file seek.
- 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 file before you can CALL it in a dBASE program.
- File write error**  
dBASE IV was unable to correctly write data to the disk.
- File write error : <filename>** [1278]  
There was a problem writing a file.
- 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.
- Find not successful** [14]  
dBASE IV was unable to find the value you searched for with the SEEK or FIND commands.

**FIRST allowed with only one half of an example variable pair** [396]

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.

**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]

Float values cannot be larger than 9.999...E+307 or smaller than .1999...E-307.

**FOR clause exceeds 220 character limit** [136]

FOR clause of an index expression cannot exceed 220 characters.

**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>** [178]

This error displays the name of the UDF called in a program or a procedure. Check to see that this function name is correct, and that the function is in the search path, or that the SET PROCEDURE or SET LIBRARY commands point to it.

**Function prohibited in SQL mode: <function name>** [340]

The function named in the error message cannot be used in SQL mode.

**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 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 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 BY, HAVING, ORDER BY, FOR UPDATE OF, or UNION 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.

**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.

**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.

**Illegal decimal length for field: <field name>** [261]

You are using CREATE FROM and the FIELD\_DEC field in the source file contains an inappropriate value for the number of decimal places for the specified field.

**Illegal field length for field: <field name>** [262]

You are using CREATE FROM and the FIELD\_LEN field in the source file contains an illegal length for the specified field.

**Illegal field name: <field name>** [224]

You are trying to use CREATE FROM and the source file contains a field name with illegal characters. Field names must begin with letters and cannot contain spaces. You can use underscores to separate words in field names.

**Illegal field type for field: <field name>** [259]

You are using CREATE FROM and the FIELD\_TYPE field in the source file contains an illegal letter for the field type of the specified field.

**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 macro expansion incorrectly.

**Illegal opcode** [68]

This is an internal error. The compiler has generated a number for a command that does not exist.

**Illegal operation. Database structure is empty or has been changed** [362]

This is a 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 use of a compiler directive** [491]

You may not use a compiler directive from the dot prompt. In a program, the defined name must begin with a letter.

**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 other commands, such as SET commands if you specify an invalid number.

**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]

You are using a dBASE function in an SQL statement, and the dBASE function does not have the correct number of input values.

**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 damaged. Do REINDEX before using data**

The index file associated with the database you are trying to USE is damaged. Reindex before using data. You can create new indexes with the INDEX 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 expression needs missing memory variable: <variable name>** [526]  
You tried to use an index whose key expression contains a variable that is undefined.

**INDEX failed - key is not DISTINCT** [236]  
The key values for your index are not unique. Select another key expression for CREATE INDEX.

**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]  
You have chosen an .ndx file during full-screen SET that does not match the database file you indicated. Choose an index file that matches the database file.

**Index interrupted. Index will be damaged if not completed** [135]  
You pressed the **Esc** key while dBASE was creating an index file. The abandoned index will not be usable.

**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. If you choose **Cancel**, an .ndx index will not be made; an .mdx index tag will be deleted from its .mdx file.

**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 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.

**Index TAG already exists: <tag name>** [205]  
The index tag you are trying to create already exists. You may replace it with the new index or cancel this command.

**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** [43]  
An operation failed because you have reached the limits of your available RAM. Try to free more RAM. Using DOS 5.0 improves the amount of available RAM.

- Insufficient memory** [1275]  
 An operation failed because you reached the limits of your available RAM. Simplify queries by breaking a long command into several shorter ones.  
 You can also free more RAM for dBASE by removing all TSRs from memory. Using DOS 5.0 improves the amount of available RAM.
- 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.
- 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 the 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.
- Integer expected** [1032]  
 A non-integer value was encountered. Replace with an integer value.
- Internal editor error. Edit buffer may be damaged.**  
 This is a fatal error that should never occur. If you get this message, please call Borland Software Support.
- 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 Borland Technical Support.
- Internal SQL error #1** [1261]  
 Undefined nodename.
- Internal SQL error #10** [1270]  
 There is an error with the optimizer join class. Simplify your query.
- Internal SQL error #11** [1271]  
 The temporary Systabl has overflowed in the optimizer. Simplify your query.
- Internal SQL error #12** [1272]  
 The temporary SYSCOLUMNS has overflowed in the optimizer. Simplify your query.
- Internal SQL error #13** [1273]  
 There is an error in temporary system catalogs in optimizer. Simplify your query.
- Internal SQL error #14** [1274]  
 The pointer to free allocated memory is invalid.
- Internal SQL error #19** [1279]  
 The internal hash table for system catalogs has overflowed. Simplify your query.

<b>Internal SQL error #2</b>	[1262]
The string table has overflowed. Simplify your query.	
<b>Internal SQL error #24</b>	[1284]
The query array has overflowed.	
<b>Internal SQL error #29</b>	[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.	
<b>Internal SQL error #3</b>	[1263]
You are using an illegal SQL command.	
<b>Internal SQL error #4</b>	[1264]
The internal relation table has overflowed. Simplify your query.	
<b>Internal SQL error #5</b>	[1265]
There is an error in the parse tree. Simplify your query.	
<b>Internal SQL error #7</b>	[1267]
There was an internal problem opening a file.	
<b>Internal SQL error #8</b>	[1268]
The internal relation table has overflowed. Simplify your query.	
<b>Internal SQL error #9</b>	[1269]
Too many columns were created in temporary relations. Simplify your query.	
<b>Internal SQL utility error #1</b>	[1218]
There was an error in the dBASE routine that returns information to DBCHECK, DBDEFINE, or RUNSTATS.	
<b>Internal SQL utility error #2</b>	[1219]
Bad file pointer encountered during execution of DBCHECK, DBDEFINE, or RUNSTATS.	
<b>Internal SQL utility error #3</b>	[1220]
Bad structure pointer encountered during execution of DBCHECK, DBDEFINE, or RUNSTATS.	
<b>Internal SQL utility error #4</b>	[1221]
Bad index pointer encountered during execution of DBCHECK, DBDEFINE, or RUNSTATS.	
<b>Internal SQL utility error #5</b>	[1222]
Bad column pointer encountered during execution of DBCHECK, DBDEFINE, or RUNSTATS.	
<b>Internal table overflow</b>	[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.	
<b>Internal: Virtual Stack Overflow</b>	[199]
This error occurs if the internal evaluation stack overflows. It can occur if an expression is too complex.	

**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 box dimensions** [227]

You have used the DEFINE BOX command with illegal values. Acceptable values for the columns are 0 through 255. Acceptable values for the height are 1 through 32767.

**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 254 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 date** [81]

You are trying to enter an invalid date into a date field. Press the **Spacebar** to clear the error message.

**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 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 APPEND FROM does not have the correct file header.

**Invalid DIF type indicator** [117]

The DIF file that you are trying to APPEND FROM contains an invalid data type indicator.

**Invalid DIF vector - DBF field mismatch** [116]

The DIF file that you are trying to APPEND FROM has an internal conflict between its header and its data.

**Invalid file extension: <.ext>**

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 file handle** [496]

The file handle you are trying to use in low-level file I/O operations is not a valid file handle number. Use the FCREATE() for a new file, and the FOPEN() for an existing file to have a valid new file handle assigned.

**Invalid file privileges** [495]

You have used the low-level file I/O functions FCREATE() or FOPEN() with an invalid file privilege letter. Use "r", "w", or "a" for read, write, or append. In combination, use "rw" for read and write, and "ra" for read append. Any other combination is invalid.

**Invalid file type for NOSAVE: <index filename>** [522]

You cannot use SET ORDER...NOSAVE with either .ndx files or the production .mdx file. You must use a non-production .mdx file.

**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, UNLOAD, or CREATE statement. Check that the filename (and path) are correctly specified. When the file type is .dbf, the extension need not be specified.

**Invalid function argument** [11]

Check the syntax for the function you are trying to use for allowable arguments.

For help on function syntax, at the dot prompt type HELP followed by the name of the function, e.g., HELP STR(). You can also press **F1** for Help. Refer to the *Language Reference* manual for complete documentation of a function.

**Invalid function name** [87]

This is a compiler error message indicating an invalid UDF function name in a program. Use the debugger to find the line of code where this function name occurs, then correct the UDF name.

**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 10. You are trying to SET ORDER TO a number that exceeds this range, or the position is empty.

**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 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 list of available programmable function keys.

**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 Lotus 1-2-3 version 2.0 spreadsheet** [297]

You are trying to import a Lotus spreadsheet that is not version 2. Please check your spreadsheet version. Convert it to a version 2 file using the Lotus conversion utility before trying to import it.

**Invalid macro file header** [356]

This error occurs if the macro file you are reading is corrupted or is not a dBASE IV macro file. It means an inconsistency was detected during the reading of the header of a macro file.

**Invalid numeric value** [497]

You are using FSEEK() with an invalid value for the third argument. This argument indicates where to start the cursor:

- 0 beginning of file
- 1 current position of pointer
- 2 end of file

**Invalid operator** [107]

The operator you are trying to use is invalid for the data types it is connecting.

**Invalid or unknown function** [31]

The function name you used in a program or procedure is unknown to dBASE IV.

**Invalid password : <table name>** [1283]

Invalid password in file header.

**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: <path>/<filename>** [202]

You have entered a non-existent or invalid filename, or given an incorrect path for the file. Check your entry.

**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.

- Invalid printer redirection** [124]  
You have redirected the print output to a printer or queue that does not exist. Check your configuration to see what ports you have.
- 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]  
The SET() function does not return a value for the keyword you used. See Chapter 4 for a list of supported keywords for SET().
- 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 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 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 table or alias name** [1172]  
You have entered an invalid name for a table or alias. Check the rules for valid names.
- Invalid TAG name** [284]  
The name you have specified for a tag contains invalid characters.
- 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.
- Keyboard macro cannot exceed 64K** [268]  
There is a 64K size limit for .key macro files.
- 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.
- Label file invalid** [54]  
The file you are trying to load onto the labels design surface is not a valid label design file (.lbl).
- Left margin plus indentation must be less than right margin** [221]  
When specifying printer output with system memory variables, the sum of the left margin and paragraph indent you used exceeded the right margin.
- 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.
- 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 100.
- 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 has been 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.

- Lower limit is larger than upper limit** [515]  
 You specified a higher value for <exp:low> than the value you specified for <exp:high>. SET KEY requires the RANGE values to be in the order low to high.
- Macro library doesn't exist** [354]  
 You used the PLAY MACRO command and no macro library is loaded. Use the LOAD MACRO command or the **Tools** menu from the Control Center to load a macro library.  
 You also get this message when using SAVE MACROS and there are no currently defined macros to save.
- 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. Use the Macros option on the **Tools** menu to check the defined macros for the currently loaded macro library.
- Macro recording is in process** [357]  
 You are trying to RESTORE MACROS while still recording a macro. Finish recording the macro first.
- Macros cannot expand flow-of-control commands** [388]  
 You cannot use macro substitution (&) to generate the following command constructs: IF...ELSE...OTHERWISE...ENDIF, DO WHILE...ENDDO, DO CASE...ENDCASE, BEGIN TRANSACTION...END TRANSACTION, ROLLBACK, START...STOP, PRINTJOB... ENDPRINTJOB, SCAN...ENDSCAN, TEXT...ENDTEXT, EXIT
- 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 number of @ GET commands exceeded** [310]  
 Reduce the number of GET commands on one page.  
 You may also increase the number of GETs allowed. To do this, adjust the GETS= statement in your Config.db file. Then reload dBASE IV.
- Maximum number of fields already reached**  
 You already have the maximum number of fields in a database file.
- 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 nested compiler directives exceeded** [490]  
 You may nest up to 32 levels of directives to the compiler for conditional compilation.
- Maximum print width of 255 characters exceeded**  
 This occurs when label width times number of labels exceeds 255 characters.
- Maximum record length exceeded** [137]  
 You are trying to create a database file whose record size is too large.

- 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.
- Maximum user count exceeded; please try again later** [499]  
The maximum number of users are logged on to the network. You should try later after some users have logged off.
- 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 field content too large** [519]  
The word wrap editor will not save more than 64K of memo text. If you need to preserve the text you already have, use **F6 Select** to select your text, then write it to a text file with the "Write/read text file" option on the **Words** menu.  
You can work with larger text fields by setting the WP setting in Config.db to use an external text editor.
- Memo fields can't be prompts** [280]  
You have attempted to ACTIVATE a POPUP which was DEFINEd with a memo field as the PROMPT FIELD <field name>.
- Memo file not found. Okay to discard links to memos?** [527]  
The .dbt file associated with the database file you tried to open cannot be found. If you choose **Yes** in response to the prompt, all connections between the .dbf file and the .dbt file will be lost. If you choose **No**, the connections will be retained and you can add back the .dbt file later.
- Memory variable already defined - cannot make PUBLIC** [271]  
You cannot change the scope of a memory variable once it is defined.
- Memory variable and dBASE function not allowed in SELECT clause 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 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.
- Memory variable not defined** [200]  
You are attempting to use a memory variable that you have not yet defined.



**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.

**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 defined** [168]

You must define a menu before you can define pads for it, activate it, or otherwise specify it or its pads.

**Menu is already in use** [181]

You have attempted to activate a MENU that is active.

**Menu screen cannot call itself, nor another menu screen** [400]

The Control Center, design screens, BROWSE, and EDIT cannot be used recursively during an interruption with DBTRAP = ON.

**Minimum field width reached**

You have tried to size a field on the layout editor smaller than is allowed for that type of field.

**Mismatched language driver for database: <filename>** [530]

The language driver assigned to the database file you tried to open is different from your current language driver. Select "Use database file" to open the file using your current language driver. This may affect how the data looks and how field names are shown and organized. However, the current language driver will not be changed, and the database file will not have the current language driver assigned to it.

**Mismatched language driver for index file: <filename>** [531]

dBASE is about to open an index file whose assigned language driver is different from your current language driver. Select "Reindex" to assign the current language driver to the index and its tags and then to reindex.

**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.

**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 ENDPRINTJOB 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 ENDPRINTJOB command before starting a new PRINTJOB. Printjobs cannot be nested.
- Missing ENDSCAN for previous SCAN command** [269]\*  
A SCAN command must be terminated by an ENDSCAN.
- 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 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 variable 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**  
You have chosen the 'Modify group' option from the Bands menu, but a non-group band is currently highlighted.
- More than one SORT option in a column** [325]  
One of your QBE columns contains more than one sort directive (e.g., Asc1, Asc2). You may only have one of these directives per column.
- Mouse is not active** [523]  
No mouse is installed.
- Must be a valid dBASE expression. Press any key to continue**  
You are using an invalid dBASE expression.
- 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 expected (cannot be reserved word) [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 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 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, 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.

**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.

**Nested function not allowed [1098]**

An aggregate function contains another nested aggregate function. Remove the nested aggregate function.

**Network server busy [148]**

Your network server is busy. Please wait until the system server can handle your request. See your system administrator.

**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 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 bars have been defined for this POPUP [166]**

Each pop-up menu must have bars defined for it.

**No current row available for UPDATE or DELETE : <cursor 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 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 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 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 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. 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 in this drive or catalog** [53]

You are doing a directory search with wildcards for a file type from the operating system, and there are no files that match the filter.

**No GROUP BY or HAVING in a nested subquery** [1167]

A nested subquery may not contain a GROUP BY or HAVING keyword.

**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 language driver specified for database: <filename> [528]**

The database file you tried to use has no language driver assigned to it. The message box offers different choices for continuing, depending on how you tried to open the file:

**Assign** — You tried to open the file in read/write mode. Select “Assign” to permanently assign your current language driver to the file and then open the file.

**View** — You tried to open the file in read-only mode. Select “View” to continue. No change will be made to the file or to your current language driver.

**Transfer Data** — You tried to APPEND FROM or otherwise copy data from the file. Select “Transfer Data” to continue. No change will be made to the file you are copying FROM. But if the file you are copying TO was created using a different language driver, the data you are copying might be displayed and organized differently in the destination file.

**No language driver specified for index file: <filename> [529]**

dBASE is about to open an index file that does not have a language driver assigned to it. If you select either “Reindex” or “Use existing index,” your current language driver will be assigned to the index and its tags. Select “Reindex” to reindex. Select “Use existing index” to use the existing index without reindexing.

**No matching #endif [494]**

You have used either #ifdef or #ifndef without a terminating #endif directive.

**No matching #ifdef or #ifndef [489]**

You have used the compiler directive #endif when there is no preceding #ifdef or #ifndef.

**No more windows available [213]**

The maximum number of available windows is 20.

**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 command [304]\***

You issued an ENDDO/ENDSCAN/ENDPRINTJOB 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 ENDPRINTJOB 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 ENDPRINTJOB 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 in structure file: <filename>** [265]  
You are attempting to CREATE FROM an empty structure file.
- 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 search string specified**  
Enter a search string before selecting **Forward/Backward search** from the **Go To** menu or pressing **Shift-F5**.
- 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.
- 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.
- Not a character expression** [45]  
The command you are using requires a character expression.  
EXAMPLE:  
KEYBOARD 'type this'
- Not a dBASE command** [83]  
The command you are attempting to use is not part of the dBASE lexicon.
- Not a dBASE database** [15]  
You are trying to USE a database file that is not in 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**  
The operating system has reported an error. dBASE IV captures the most common operating system errors and provides you with a dBASE IV version. In this case, the error was a rare type of operating system error that dBASE IV does not rephrase.

- 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 directory: <path>** [413]  
Either the path does not exist or there is a spelling error.
- Not a valid disk drive: <drive>** [412]  
The drive you are trying to access is not recognized by the operating system.
- Not a valid expression** [233]  
You have used an invalid dBASE IV expression in a program. Find the exact program line in the debugger and correct the syntax.
- 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 use an invalid .qbe or .qry file. Create a new .qbe file in dBASE IV. .qry files are created in dBASE III PLUS, and cannot be created in dBASE IV. Use dBASE III PLUS to create a new .qry 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 attempted to use an invalid .vue file. .vue files are created in dBASE III PLUS, and cannot be created in dBASE IV. Use dBASE III PLUS to create a new .vue file.
- Not a valid window file** [358]  
You used RESTORE WINDOW, and the specified source file is not a window file.
- 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**  
The operating system reports that it has run out of disk space for the current operation.
- 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 disk, or specify another drive.
- 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 when one of the search options on the **Go To** menu for **Browse** or **Edit** finds no data that matches the search criteria.

**Not suspended****[101]**

You issued a **RESUME** command when the file execution was not suspended.

**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 **Colcount** 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 variables exceeds number of fields** [410]

You have specified too many memory variables using AVERAGE, SUM, or CALCULATE with the TO phrase.

**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.

**Numbers are not allowed in the CURRENCY symbol** [295]

Use only characters to define a currency symbol.

**Numeric overflow (data may have been lost)** [39]

You have tried to REPLACE a large number into a field that cannot hold the complete number. If the number can be stored in exponential form, then as many significant digits as possible have been stored in the field. If the number is too large to store this way, then the field was filled with asterisks.

**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 1 to 2 End Of Line characters may be defined** [498]

You may define a maximum of 2 end of line characters with the FGETS() low level I/O function. These are usually CHR(13) and CHR(10).

**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.

**Only one sort or GROUP BY if on a calculated field** [506]

There is a sort or a GROUP BY in a calculated field column and there is another sort or GROUP BY in another column somewhere on the QBE design surface.

**Only one sort or GROUP BY if on a complex index** [507]

There is a sort or a GROUP BY in a complex index and there is another sort or GROUP BY in another column somewhere on the QBE design surface.

**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 have either:

1. Attempted to SORT or INDEX on a calculated field.
2. Placed an operator in the pot handle of the file skeleton for calculated fields in QBE.

**Operation not allowed in an SQL file** [149]

Check the syntax of your command line.

**Operation not allowed with a user-defined function** [150]

There are some procedures that cannot contain user-defined functions.

For example, with DBTRAP ON, you cannot put a user-defined function into an index key expression.

EXAMPLE:

```
INDEX ON Myudf(Lastname) TO Test150
```

**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 or you are trying to join memo fields.

**Operations not allowed in field column** [513]

There is an operation, REPLACE, APPEND, UNIQUE, FIND, in a field column of a file skeleton. Operations are only allowed in the filename column.

**Operator not allowed for 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 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 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.

**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.

**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.

**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.

**PAD has not been defined [164]**

You must define a menu pad before you can specify it in a command or function.

**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.

**Path too long [1044]**

An operating system path name may not exceed 64 characters.

**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.

**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 has less than one row, and is too small to have any selection bars. Define a larger popup and selection bars.
- POPUP not DEFINED** [165]  
You must define a pop-up menu before you can define bars for it, activate it, or otherwise specify it or its bars.
- 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.
- 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.
- Positive value required** [525]  
This function requires a number greater than zero.
- 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 ENDPRINTJOB command before you can start another printjob. Issue an ENDPRINTJOB command to clear this error.
- PROCEDURE command is required** [242]  
dBASE procedures must begin with the command word PROCEDURE.
- Procedure is too large (codesize > 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]  
You have used RETURN <value> in a PROCEDURE. The RETURN command can take a value when used in a FUNCTION, but not in a PROCEDURE.
- PROCEDUREs/FUNCTIONs nested too deep** [67]  
You do not have enough available memory for the number of levels you are attempting to nest your commands.

**Production .MDX file not found: <filename>** [210]

The production .mdx file for the database in use does not exist or is not in the same directory as the database. If you choose to Proceed, the database file will not longer look for the .mdx file. You can create a new .mdx with the INDEX command. This error can also occur if you renamed the files. The database file cannot find the .mdx file because it is looking for an .mdx file with its original name.

**Production MDX file is damaged** [289]

If you reindex, a new .mdx file will be created. If you proceed without reindexing, the index will include damaged information.

**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.

**PROTECT cannot be used on SQL tables**

SQL tables must be protected with GRANT/REVOKE. You cannot encrypt a SQL table with PROTECT.

**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>**

The operating system has detected a read problem on the indicated drive.

**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]

The record you want to use is locked by the user identified in the error message.

**Record in use by another** [109]

The record you want to use is currently in use 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 began to INSERT a record, but abandoned the action. This message confirms that the record was not inserted.
- Record out of range** [5]  
You are trying to go to a record that does not exist.
- Relation record in use by another** [142]  
The active database file is related to another file that is currently being used 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 254 characters. You will also get this message if the second argument is -1 or less.
- 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 bracket expected** [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 margin must be less than or equal to 255** [225]  
The right margin setting cannot be greater than the page width.
- 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.
- Rollback database cannot be executed inside a transaction** [197]  
You cannot apply the ROLLBACK command to a specific database file during a transaction, e.g., 'ROLLBACK Employee'. Use 'ROLLBACK' during a transaction.
- Row 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.
- 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.

**Screen does not exist: <screen filename>** [379]

You are using RESTORE SCREEN. The file that you indicated would contain the saved screen does not exist.

**Screen file invalid** [283]

The screen design file (.scr) has been corrupted. Although you may modify the generated code file (.fmt) with a text editor, you should only change the .scr file on the forms design screen.

**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 box with F6**

This is a Control Center error. You see this message when you try to use navigation keys during an extended selection of a box.

**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 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.

**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 may press the A or D key to choose one of the acceptable ascending or descending sort orders for the 'Sort database on field list' option of the **Organize** menu. This message appears when you press a letter key besides A or D.

**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 index cannot contain more than 10 columns** [1173]

You cannot create a SQL index with more than 10 column references.

- SQL run-time error** [335]  
This message is displayed if an error occurs during the execution of an SQL statement.
- 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.
- 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 characters.
- STR() : Out of range** [63]  
This error occurs when the third argument of the STR() function is equal to or greater than the second argument.
- 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 created by STUFF() is longer than 254 characters.
- 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.
- SUBSTR() : Length value out of range** [40]  
The length value (the third argument) cannot be a negative number.
- 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]  
The syntax of the last command you attempted to issue was incorrect. This can occur in programs or from the dot prompt.  
For help on command syntax at the dot prompt, type HELP followed by the name of the command, e.g., HELP BROWSE. You can also press **F1** for Help. Refer to the *Language Reference* manual for complete documentation of a command.
- 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 FOR clause** [404]  
The FOR clause requires a condition such as a comparison between two or more items or a logical statement.

**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.

**System is not configured for current code-page** [162]  
The DOS code page parameters are in conflict with the language version of dBASE IV. Verify that the COUNTRY and DEVICE settings in CONFIG.SYS match the language version of dBASE IV you are using. Also check the NLSFUNC, KEYB, and MODE CODEPAGE PREPARE settings in AUTOEXEC.BAT.

**System table entry missing for table: <table name>** [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.

**Tab stops must be in ascending order** [226]  
The values assigned to the system memory variable \_tabs must be given in ascending order.

**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 full** [105]  
You can load a maximum of 16 binary files into the CALL table which you can then call from within dBASE IV. Reduce the number of files.

**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.

**Table(s) not DBDEFINEd:**  
Lists files which were not converted to SQL tables. See the DBDEFINE completed; errors/warnings found error message.

**Tag expression not in range** [517]

You have referred to an index with a number that is not within the acceptable range. The number cannot be less than 1 or greater than the number of tags in the index. An .mdx file can have between 1 and 47 tags.

**TAG not found: <tag name>** [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.

**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 SYSTABLES: <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 file cannot be modified** [342]

This message indicates that the file on which the cursor is positioned in the Control Center cannot be modified when the design key (**Shift-F2**) is pressed. Files with file extensions of .qbo, .fmo, .fro, .lbo, .dbo, .exe, and .com cannot be modified.

**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 @ 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 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 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 authorized to you. See your system manager.
- 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 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 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 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 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).
- 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.
- 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.

- Unable to load Command.com** [92]  
 You are trying to load another copy of the DOS command interpreter Command.com to run an external program, or a batch file, but dBASE is unable to load another level of the command interpreter. This is because you do not have enough free RAM.
- Unable to load file**  
 The design file you specified for a form, report, or label does not match the design surface you are trying to use, or the file you specified is corrupted.
- Unable to LOCK** [129]  
 The record you want to lock is already in use by another user.
- 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.
- Unable to SKIP** [128]  
 The record pointer is at the last record in the file or at the limit of the scope modifier.
- Unassigned file number** [2]  
 This message can occur if you attempt an I/O operation by using a file handle number that hasn't been opened. This is an internal error and should not occur under normal dBASE IV usage.
- Unauthorized access level** [133]  
 You are trying to access a file which you are not authorized to access. Please see your system administrator about access levels.
- Unauthorized login** [132]  
 You are trying to log in without a valid password. See your system administrator for a valid password assignment.
- Unbalanced parenthesis** [8]  
 You must have an equal number of opening and closing parenthesis.
- 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 Sysstns catalog table.) Check that you have not given a table or view name by mistake. Use a SELECT operation on the Sysstns 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, Sysstns, 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.

**Unknown command code: <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.

**Unknown compiler directive: #<compiler directive>** [493]

The compiler directive displayed in the error message is not supported in dBASE IV. You cannot use it in your code.

**Unknown function key** [104]

You specified a function key with a number, and the number is not within the allowed range of 2 through 29.

EXAMPLE:

```
SET FUNCTION 29 TO "This is 29" && Works
```

```
SET FUNCTION 30 TO "This is 30" && Error
```

**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. Remove the beginning underscore character from your memory variable name.

**Unlink of old name incomplete, errno =** [84]

You may not rename a file that is in a relationship chain with other database files. If you are sure you want to rename this database file, remove it from the relationship, then rename it.

**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]

You specified a string expression without a valid terminating delimiter. The valid delimiters are single quotes ( ' '), double quotes ( " "), and 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 start a new transaction. If there is no transaction in progress, you may have to RESET the database file header.

**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 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.

**User name has not been defined** [156]

The PROTECT utility does not contain the name you are trying to use. See your system administrator.

**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.

**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.

**Variable names have a maximum length of 10** [312]

You entered a variable name longer than 10 characters in a QBE file skeleton.

**Variable not found: <variable name>** [12]

You used a name in a command or function that is not the name of an active field or memory variable.

**View cannot hold any more fields (255 already)** [353]

You tried to add a 256th field to the view.

**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**

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**

This warning 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**

This warning 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.

**Warning on line <program line number> <warning message>** [97]

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: 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: Uncompleted transaction found** [190]  
An existing transaction log file was detected on your hard disk by dBASE IV when it was loaded. Use the `ROLLBACK` command to restore your data.  
Although you cannot use `ON ERROR` to trap this situation, you can use `ERROR()` to detect it right after dBASE IV has loaded.
- WINDOW coordinate(s) outside of allowable screen space** [332]  
You are trying to `ACTIVATE` a WINDOW whose bottom border is too low for the current display mode. Either redefine the window or change the display mode to one with more available rows.
- 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 short to be used by `BROWSE`. The window must be able to display at least one row of data. Try making the window taller or using the `COMPRESS` keyword with `BROWSE`.
- 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: <filename>/<alias>** [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 <disk>**  
This is an operating system error indicating a write error on the disk identified in the message.
- Wrong number of parameters** [94]  
You are trying to `CALL` a binary procedure file with more than the maximum seven parameters.



**Zoom window command editor, CTRL-END exits to dot prompt [403]**

This is an information message from the zoom window command editor.

**^ or \*\*: Negative base, fractional exponent [78]**

You cannot raise a negative number to a fractional power. In other words,  $(-1)^{.5}$  cannot be handled by dBASE IV. This would require the introduction of imaginary number arithmetic.

**^--- 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.

**^--- Out of range [75]**

The value specified for the indicated setting in your Config.db file is not within the allowed range of values for that setting.

For example, 'PRECISION = 5' would not be allowed.

This error may also be returned when invalid values are assigned to system memory variables.

**^--- Truncated [74]**

A line of text in the Config.db file was too long or unterminated, and was truncated. The maximum allowed string length for Config.db is 255.

**^^ Expected ON or OFF [73]**

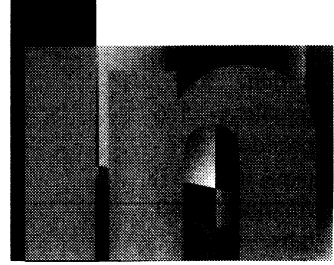
The indicated setting from your Config.db file should have ON or OFF after the equals sign.

EXAMPLE:

TALK = ON



# Index



## Symbols

- \$ operator
  - for substring comparison, 18
- & (macro substitution), 389
- \* symbol, 640, 649
- .lbo files, 106
- .mdx files and CATALOGs, 292
- ? and ?? difference, 29
- ? command
  - alignment functions, 31
  - AT clause, 31
  - continuation character, 29
  - currency display functions, 31
  - font control, 31
  - format functions of, 30
  - leading zero display in, 31
  - output appearance, 30
  - output redirection with, 29
  - STYLE clause, 31
  - with ON commands, 31
  - word wrap character of, 30
- ???, 33
- ??? command, 33
- @
  - \$ function for row and col, 38
  - \$ template for currency format, 41
  - COLOR, 38
  - combining format functions in, 42
  - currency symbol in, 42
  - DEFAULT, 39
  - ERROR message text in, 39
  - format function precedence, 43
  - Format functions of, 41
  - FUNCTION option in, 39
  - MESSAGE option, 39
  - numeric templates in, 44
  - PICTURE option, 40
  - RANGE option, 40
  - REQUIRED, 41
  - template symbols in, 43
  - VALID option, 40
  - WINDOW option for GETs, 39
- @ coordinates, 37
- @...CLEAR
  - to erase parts of display, 46
- @...GET, 38
- @...SAY, 38
- @...SCROLL, 48
- @...CLEAR, 46
- @...FILL, 47
- @...SCROLL, 48
- @...TO, 49
- \_box, 556
- \_dbaselock field
  - and COPY STRUCTURE, 95
  - EXTENDED, 96
  - and multiuser lock detection, 87
- \_indent, 557
- \_lmargin, 558
- \_padvance, 559
- \_pageno, 561
- \_pbpage, 562
- \_pcolno, 564
- \_pcopies, 565
- \_pdriver, 566
- \_pecode, 568
- \_peject, 569

- \_pepage, 570
- \_pform, 571
- \_plength, 572
- \_ploffset, 574
- \_ppitch, 575
- \_pquality, 576
- \_pscode, 577
- \_pspacing, 578
- \_rmargin, 580
- \_tabs, 582
- \_wrap, 583

## A

### Abbreviation

- of commands, 7
- of keywords, 7

ABS(), 390

### ACCEPT

- prompt limits, 51

Access level, 391

ACCESS(), 391

### Accessing

- contents of the \_dbaselock field, 477
- the CATALOG file, 404

ACOS(), 393

ACTIVATE MENU, 52

ACTIVATE POPUP, 53

ACTIVATE SCREEN, 53

ACTIVATE WINDOW, 54

Active index count, 537

Aggregate functions, 75

- in SELECT clause, 640, 641

Alias (SQL)

- defining, 641

Alias names, 277

- assigned by dBASE IV, 278

- in SQL SELECT clause, 641

ALIAS(), 394

### Aliases

- default, 10

- invalid, 10

- SQL, defining, 641

### alignment

- of ??? output, 555

ALL keyword (SQL)

- in GRANT, 626

- in REVOKE, 635

- in SELECT, 640

Alt key combinations (table), 462

ALTER privilege (SQL), 626

ALTER TABLE (SQL), 592, 638

APPEND, 55

- file types in, 57

- replacing values, 155

APPEND FROM, 56

APPEND FROM ARRAY, 59

APPEND MEMO, 61

Applications Generator Objects, 104

Array defined, 14

### Arrays

- as memory variables, 120

- dimensions, 679

- total elements, 679

ASC(), 394

ASCII Chart, 699

ASCII decimal code, 394

### ASCII files

- evaluating ASCII values, 643

- exporting from SQL to, 660

- importing to SQL from, 631

ASIN(), 395

Assembly program preparation, 186

ASSIST, 62

AT(), 396

ATAN(), 397

ATN2(), 397

Automatic compilation, 316

Automatic locking commands, 347, 348

Available disk space determination, 421

Available memory, 178

AVERAGE, 63

## B

Backward compatibility  
  of binary files, 185  
  of file formats, 694  
  of file structures, 96  
  of format files, 330  
  of index files, 276  
  of labels, 107  
  of memo fields, 90  
  of numeric fields, 17  
  of object code files, 83  
  of procedure files, 138  
  of queries and views, 108, 384  
  of query files, 328  
  of report forms, 110  
  of reports, 243  
  of returned filenames, 331  
  of screen files, 112  
  of SET commands, 20  
  of view files, 113

BAR(), 398  
BARCOUNT(), 399  
BARPROMPT(), 400  
BEGIN/END TRANSACTION, 64  
  with DELETE, 617  
  with ROLLBACK, 637  
Beginning of file, 401  
Binary Named List  
  creating, 134  
Black and white on color monitors, 297  
BLANK, 68  
  testing for, 68  
Blank formats, 68  
Blanking fields, 69  
Blanks, testing for, 464  
Blink attribute, 299  
Blocking access to design mode, 316  
BNL file extensions, 134  
BOF(), 401  
Box drawing, 49  
Box printing, 50  
Branching  
  on error condition, 214  
  on ON ERROR, 195  
  on ON ESCAPE, 195

  on search results, 447  
Breakpoints, 476  
BROWSE, 70  
  calculated fields read-only, 71  
  COMPRESS, 72  
  field width limits, 72  
  format files, 72  
  LOCK, 72  
  NOAPPEND, 72  
  NOCLEAR, 72  
  NODELETE, 72  
  NOEDIT, 72  
  NOFOLLOW, 72  
  NOINIT, 71  
  NOMENU, 72  
  NOORGANIZE, 72  
  restricting fields, 73  
  special syntax notation in, 70  
BROWSE to EDIT, 71  
BROWSE to QBE, 71

## C

CALCULATE, 75  
CALL, 77  
  LOAD, 77  
  parameters, 78  
CALL(), 403  
Calling binary programs, 77  
Calling the debugger, 380  
CANCEL, 78  
Carrying data to new record, 290  
Catalog, dBASE  
  effect of CREATE TABLE, 602  
  effect of DROP DATABASE, 619  
  effect of DROP TABLE, 621  
  effect of DROP VIEW, 621  
  effect of SAVE TO TEMP, 647  
  effect of start DATABASE, 657  
Catalog file structure, 292  
Catalog tables, 667  
  described, 670  
  displaying information, 668  
  object definitions, 668  
  updating statistics, 668

- Catalog tables (SQL)
  - creating entries, 596, 610
  - effect of DROP DATABASE, 619
  - effect of DROP TABLE, 621
  - statistics, 638
  - tracking privileges, 626
  - verifying entries, 609
- Catalog title prompt, 379
- CATALOG(), 404
  - with SET FULLPATH, 404
- CDOW(), 405
- CEILING(), 405
- CERROR(), 406
- CHANGE, 79. *See also* EDIT
- CHANGE(), 407
- Changing the controlling index, 277
- Changing the decimal separator, 360
- Changing the separator character, 373
- Character data type, 16
- Character expression comparison, 484, 491
- Character string functions
  - DIFFERENCE(), 420
  - ISLOWER(), 467
  - LEN(), 474
  - LIKE(), 475
  - STR(), 534
- Character strings
  - extracting substrings, 536
  - removing blanks, 520
  - replacing part, 535
- Character to date conversion, 413
- Child file, 367
- Choosing work areas, 258
- CHR(), 408
- Clause (SQL), 646, 647
  - in UNION query, 646
- CLEAR, 80
  - ALL, 80
  - FIELDS, 80
  - GETS, 80
  - MEMORY, 80
  - MENUS, 81
  - Options, 80
  - POPUPS, 81
  - SCREENS, 81
  - TYPEAHEAD, 81
  - WINDOWS, 81
- Clock display format, 336
- Clock screen position, 295
- CLOSE, 81
- CLOSE (SQL), 594, 623
- Closing a catalog, 292
- Closing low-level files, 444
- CMONTH(), 410
- code page, 344
- COL(), 411
- COL() compared to \$, 411
- Color attribute letters, 296, 299
- Color settings of windows, 303
- Column headings
  - in LIST /DISPLAY, 176
- Column width of memo fields, 352
- Columns
  - adding, 592
  - changing values, 661
  - in a view, 608
  - initializing values, 628
  - INSERTing values, 628
  - name qualification, 640
  - naming, 602
  - removing, 593
  - restrictions, 593
  - SELECT headings, 641
  - SELECTing, 640
- Command history buffer, 7
- Command line
  - editing window, 6
  - length, 682
- Commands (SQL)
  - allowed in transaction, 637
  - classes of, 589
  - data definition, 590
  - database creation, 589
  - database security, 589
  - database start-up, 589
  - deletion, 590
  - embedded, 591
  - object creation, 589
  - object modification, 589
  - query update, 591

- syntax, 587
  - utility, 591
- Comment lines, 193
  - continuation character, 193
- Comparison operators (SQL), 642
- COMPILE, 82
  - code optimization, 86
  - optimization, 86
- COMPILE compared to DO, 83
- Compiled macros, 84
- Compiler errors, trapping, 406
- COMPLETED(), 412
- Condition defined, 14
- Conditional append from array, 60
- Conditional compilation, 85
- Conditional copying to an array, 98
- Conditional indexing, 164, 445
- Conditional searches, 188
- Config.db file, 597
  - changing date format, 17
- Constant, 15
  - in SQL SELECT clause, 640
- Continuation character, 11
  - in comment lines, 193
- CONTINUE, 86
- Control Center, 62
- Control character specifiers, 34
- Controlling index, 163
  - changing, 356
- Controlling index tag, 487
- Controlling mdx, 164
- Conversion functions
  - & (macro substitution), 389
  - ASC(), 394
  - CHR(), 408
  - CTOD(), 413
  - DMY(), 421
  - DTOC(), 423
  - DTOR(), 424
  - DTOS(), 425
  - FIXED(), 437
  - FLOAT(), 441
  - LOWER(), 482
  - MDY(), 487
  - RTOD(), 519
  - STR(), 534
  - TRANSFORM(), 541
  - UPPER(), 545
  - VAL(), 547
- Conversion of field data types, 102
- CONVERT, 87
- Converting
  - date format, 487
  - fields, numeric to characters, 102
  - lower to uppercase, 545
  - .mdx to .ndx, 96
  - .ndx to .mdx, 92
  - number to string, 534
  - radians to degrees, 519
  - to numeric expression, 547
  - upper to lowercase, 482
- COPY, 88
  - and \_dbase lock field, 89
  - with SET RELATION, 90
- COPY FILE, 91
- COPY INDEXES, 92
- COPY MEMO, 93
- COPY STRUCTURE, 94
- COPY STRUCTURE EXTENDED, 95
- COPY TAG, 96
- COPY TO ARRAY, 97
- Copying
  - fields into new record, 290
  - production .mdx file, 89
- Correlated Subquery (SQL), 641, 644, 651
- COS(), 413
- COUNT, 99
- CREATE DATABASE (SQL), 596, 657, 672
- CREATE FROM, 103
- CREATE INDEX (SQL), 598, 639, 668, 672
- CREATE or MODIFY STRUCTURE, 100
- CREATE SYNONYM (SQL), 600
- CREATE TABLE (SQL), 602
- CREATE VIEW (SQL), 605, 651
- CREATE VIEW FROM ENVIRONMENT, 113
- CREATE/MODIFY APPLICATION, 104
- CREATE/MODIFY LABEL, 106

CREATE/MODIFY QUERY VIEW, 107  
CREATE/MODIFY REPORT, 109  
CREATE/MODIFY SCREEN, 111

Creating

- a CATALOG, 291
- a database, 100
  - from the fields list, 326
  - with JOIN, 170
  - with SORT, 262
  - with TOTAL, 270
- a database file
  - in programs, 103
- an empty database file, 94
- arrays, 119
- format files, 111
- reports, 243

CTOD(), 413

Currency symbol, position of, 306

Cursor (SQL)

- activating, 633
- deactivating, 594
- defining, 612
- positioning, 622
- referencing, 616
- using to delete, 616, 617
- using to update, 647, 663

Cursor control keys

- in EDIT, 145

## D

Data encryption, 230

Data types, 16

- allowed in filters, 328

Data types (SQL), 592, 603, 643

Database

- activating, 657
- closing, 657, 658
- default, 657
- displaying information, 656

Database (SQL)

- creating, 596
- deleting, 619
- naming, 596

Database directory (SQL), creating, 596

Database field descriptor bytes, 696

Database file

- importing to SQL from, 631
- SQL table as, 602

Database file functions

- BOF(), 401
- DELETED(), 417
- EOF(), 425
- FOUND(), 447
- LOOKUP(), 481
- LUPDATE(), 483
- MEMLINES(), 488
- MLINE(), 492
- RECCOUNT(), 512
- RECNO(), 513
- RECSIZE(), 514
- SEEK(), 523

Database file specifications, 679

Database file structure, 695

Database records

- in .dbf file structure, 696

Date

- day of week, 422
- determining earlier, 491
- determining year, 550
- of UPDATE, 483

Date comparison, 484, 491

Date delimiter character, 350

Date delimiters, 17

Date display formats, 308

Date format conversion, 421

Date formats, 308

Date functions

- CDOW(), 405
- CMONTH(), 410
- CTOD(), 414
- DATE(), 414
- DAY(), 415
- DMY(), 422
- DOW(), 422
- DTOC(), 423
- DTOS(), 425
- MDY(), 487
- YEAR(), 550



- Date to character conversion, 423, 425
- Date type, 17
- DATE(), 414
- DAY(), 415
- dBASE commands allowed in SQL, 705
- dBASE functions allowed in SQL, 707
- dBASE II files
  - exporting from SQL to, 660
  - importing to SQL from, 631
- dBASE III files and dBASE III PLUS files
  - exporting from SQL to, 660
- DBASE.ERR file, 709
- DBCHECK (SQL), 609
- DBDEFINE (SQL), 610, 648, 669
- DBF(), 416
- DBTRAP protection, 309
  - and LIST/DISPLAY, 310
  - and recursive BROWSE, 310
- DEACTIVATE MENU, 114
- DEACTIVATE POPUP, 115
- DEACTIVATE WINDOW, 116
- DEBUG, 117
  - breakpoint window, 117
  - commands, 117
  - debug window, 117
  - display window, 117
  - edit window, 117
  - parameter list, 118
  - windows, 117
- DEBUGGER
  - next command, 118
  - program line, 118
  - program trace, 118
  - stepping through commands, 118
- Decimal places displayed, 311
- DECLARE, 119
- DECLARE CURSOR (SQL), 594, 612, 616, 622, 633, 647, 652
- Default border, 289
  - with @ command, 289
- Default drive
  - with SET DEFAULT, 312
  - with SET DIRECTORY, 312
- default field delimiter character, 314
- DEFINE BAR, 122
- DEFINE BOX, 123
- DEFINE MENU, 125
- DEFINE PAD, 127
- DEFINE POPUP, 129
- DEFINE WINDOW, 130
- Defining UDFs, 153
- Degrees to radians conversion, 424
- DELETE, 132
- DELETE (SQL), 639, 651
  - embedded form, 616, 653
  - using cursor, 615
  - with Where clause, 616
- DELETE and the record pointer, 132
- DELETE FILE, 133
- DELETE privilege, 626
- DELETE TAG, 133
- DELETED(), 417
- Deleting
  - all records, 280
  - fields, 102
  - files, 148
  - marked records, 132
  - memory variables, 236
  - records, 219
- Delimiters, 16
- DESCENDING(), 418
- Determining
  - controlling index, 498
  - printhead position, 502
- DEXPORT, 134
- DGEN(), 419
- DIFFERENCE(), 420
- DIR, 135
- Directory information, 177
- DISPLAY, 136
- Displaying
  - boxes, 556
  - century prefixes, 294
  - column headings, 333
  - command lines, 320
  - contents of ASCII files, 272
- DISTINCT keyword (SQL), 640
- DMY(), 421
- DO, 137
  - search order, 137
  - with update queries, 140

- DO CASE, 141
  - vs. IF, 161
- DO WHILE, 142
- DOS I/O error codes, 432
- DOS programs that TSR, 252
- Dot prompt
  - signals interactive mode, 6
- DOW(), 422
- DROP DATABASE (SQL), 618
- DROP INDEX (SQL), 619
- DROP SYNONYM (SQL), 620
- DROP TABLE (SQL), 620
- DROP VIEW (SQL), 621
- DTOC(), 423
- DTOR(), 424
- DTOS(), 425

## E

- ECHO to printer with DEBUG, 311
- EDIT, 144
  - NOAPPEND options, 145
  - NOCLEAR options, 145
  - NODELETE options, 145
  - NOEDIT options, 145
  - NOFOLLOW options, 145
  - NOINIT options, 145
  - NOMENU options, 145
- EDIT to BROWSE, 144
- editing the alternate file, 284
- EJECT, 147
- EJECT compared to EJECT PAGE, 148
- EJECT PAGE, 147
- Encrypted files, 321
- Encryption, dBASE
  - effect on DBCHECK, 609
  - use of DBDEFINE, 611
- Encryption, SQL
  - use of DBDEFINE, 611
- Environment variable checking, 455
- EOF(), 425
- ERASE, 148
- ERASE and ? clause, 149
- ERRCODE.TXT file, 709
- Error message numbers, 426

- Error message text, 491
- Error messages, 491, 709
- Error recovery, 248
- Error trapping, 213
- ERROR(), 426
- errors
  - trappable, 709
  - unrecoverable, 709
- Evaluating data types, 543
- Exclusive use, 324
  - required with ZAP, 280
- EXP(), 427
- Explicit file locking, 442
- Explicit locking and unlocking, 516
- Explicit work area expression, 259
- Exponentiation function, 427
- EXPORT, 149
- Export file types (SQL), 660
- Exporting data with COPY, 90
- Exporting SQL data, 660
- Expressions, 15
  - in SELECT clause, 640, 641
  - in SQL search conditions, 642
- Extracting characters, 536
- Extracting text from memo field, 492

## F

- FCLOSE(), 428
- FCREATE(), 429
- FDATE(), 430
- FEOF(), 431
- FERROR(), 432
- FETCH (SQL), 595, 613, 616, 622, 633
- FFLUSH(), 433
- FGETS(), 434
- Field list
  - scanning contents, 440
- Field list from multiple files, 325
- Field list ignoring commands, 326
- Field list in multiple work areas, 325
- Field sizes, 679
  - limits in exported files, 150
- FIELD(), 435

- Fields
  - as production .mdx tags, 101
  - in another work area, 15
- File extensions, 691
- File integrity tag, 244
- File pointer in low-level files, 431
- File relationships, 694
- FILE(), 436
- Files
  - date of update, 483
  - determining existence, 437
  - determining size, 452
  - option of pop-up, 130
  - time last modified, 452
- Filter condition, 328
- Filtering with SET FIELDS, 324
- Filters
  - and JOIN, 170
  - in REPLACE FROM ARRAY, 241
- Financial functions
  - FV(), 453
  - PV(), 508
- FIND, 150
  - and related files, 151
  - compared to LOCATE, 150
  - compared to SEEK, 150
- Finding words that sound alike, 532
- FIXED(), 437
- FKLABEL(), 438
- FKMAX(), 438
- FLDCOUNT(), 439
- FLDLIST(), 440
- FLOAT(), 441
- FLOCK(), 442
- FLOOR(), 443
- FOPEN(), 444
- FOR defined, 14
- FOR UPDATE OF Clause, 613, 647, 648, 649
- FOR(), 445
- Format file, 45, 111, 329
  - compiled version of, 330
  - interference in READING GETs, 330
  - modifying, 330
  - sections of, 330
- FOUND(), 447
- FPUTS(), 448
- Framework II files
  - exporting from SQL to, 660
  - importing to SQL from, 631
- FREAD(), 449
- FROM Clause (SQL), 641, 642
- FSEEK(), 451
- FSIZE(), 452
- FTIME(), 452
- FUNCTION, 153
- Function key assignments, 332
- Function key labels, 438
- Functions, dBASE, 389
  - in INSERT, 628
  - in search conditions, 642, 644
  - in SELECT clause, 640
- Functions defined, 20
- Future value, 453
- FV(), 453
- FWRITE(), 454

## G

- Generic.pr2 driver, 567
- GETENV(), 455
- GO/GOTO, 157
  - in related files, 158
  - restoring record pointer, 155
- GRANT (SQL), 624, 635, 671
- GROUP BY Clause, 644, 649
  - use of index, 598
- Group result, 644

## H

- HAVING Clause (SQL), 644, 651
- Header structure of .dbf files, 695
- HELP, 159
- Hiding password displays, 299
- Hiding the cursor, 307
- History buffer, 335
- History buffer contents, 178
- HOME(), 456

# I

- ID(), 457
- Identification functions
  - ALIAS(), 394
  - BAR(), 398
  - DBF(), 416
  - FIELD(), 435
  - FILE(), 436
  - FKLABEL(), 438
  - FKMAX(), 438
  - GETENV(), 455
  - MDX(), 486, 496
  - NDX(), 496
  - NETWORK(), 497
  - ORDER(), 498
  - OS(), 499
  - PAD(), 499
  - PROGRAM(), 506
  - PROMPT(), 506
  - SET(), 526
  - TAG(), 537
  - VARREAD(), 548
  - VERSION(), 549
- Identifying network users, 184
- IF vs. DO CASE, 161
- IF...ELSE with DELETE, 617
- IF/ENDIF, 160
- IIF(), 457
- Immediate IF, 457
- IMPORT, 161
  - supported formats, 161
- Import file types (SQL), 631
- Importing data (SQL), 630
- INDEX, 162
- Index (SQL)
  - automatic operation, 599
  - catalog entry, 667, 672
  - creating, 598
  - deleting, 619
  - naming, 598
  - restrictions, 599
  - unique, 599
  - updating statistics, 638
- Index, dBASE
  - unique, 544, 611
- Index functions
  - DESCENDING(), 418
  - FOR(), 446
  - MDX(), 486
  - NDX(), 496
  - TAG(), 537
  - TAGCOUNT(), 538
  - TAGNO(), 538
  - UNIQUE(), 545
- INDEX privilege (SQL), 626
- Index tag (SQL), 599
- Indexing
  - and filters, 165
  - finding KEY expressions, 469
  - .mdx file updates, 163
  - on a UDF, 310
  - on dates, 425
  - on memory variables, 165
  - removing UNIQUE status, 382
  - with both .mdx and .ndx, 357
  - with UDFs, 156
- Indirect file references, 8
  - for macros, 390
- Indirect reference to filename, 276
- INKEY(), 459
  - numeric argument, 459
  - returned values, 460
- INPUT, 168
  - compared to ACCEPT, 168
- Input functions
  - INKEY(), 459
- INSERT, 169
- INSERT (SQL), 627, 638, 648
- INSERT privilege (SQL), 626
- Inserting fields into a database structure, 102
- INT(), 463
- Interest rate, 508
- Internal decimal separator, 360
- Interrupt handling and DBTRAP, 309
- Interrupting program execution, 78, 322
- INTO Clause (SQL), 641, 651
- ISALPHA(), 464
- ISBLANK(), 464
- ISCOLOR(), 466
- ISLOWER(), 467

ISMARKED(), 467  
ISMOUSE(), 468  
ISUPPER(), 469

## J

JOIN, 170  
Join (SQL), 649

## K

KEEP option (SQL), 648  
Key label mnemonics, 173  
Key label names  
    of non-printing keys, 197  
KEY(), 469  
KEYBOARD, 172  
Keyboard macros, 221  
Keyboard input in programs, 173  
Keyboard macro execution, 221  
KEYMATCH(), 470

## L

Label design screen, 175  
Label designer, 106  
Label files, 174  
LABEL FORM, 174  
    file locking, 175  
Language components, 5  
language driver compatibility, 344  
language tables, 344  
LASTKEY(), 472. *See also* INKEY()  
Left margin  
    of ??? output, 558  
    of printed output, 349  
LEFT(), 473  
LEN(), 474  
LIKE(), 475  
Line numbers  
    in programs, 476  
    of streaming output, 573  
LINENO(), 476  
Linking database files, 374

LIST compared to DISPLAY, 175  
LIST/DISPLAY, 175  
LIST/DISPLAY FILES, 177  
LIST/DISPLAY HISTORY, 178  
LIST/DISPLAY MEMORY, 178  
LIST/DISPLAY STATUS, 181  
LIST/DISPLAY STRUCTURE, 182  
LIST/DISPLAY USERS, 184  
Literal text strings in programs, 269  
LKSYS(), 477  
LOAD, 184  
LOAD DATA (SQL), 630  
Loading binary programs, 184  
Loan payment amortization, 501  
LOCATE, 188  
LOCATE and CONTINUE, 189  
Lock attempts, 371  
LOCK(), 479. *See also* RLOCK()  
Locked records and screen refresh, 366  
Locking  
    files, 442  
    records, 517  
Locking information, 477  
LOG(), 479  
LOG10(), 480  
Logarithm, 428  
    common, 480  
    natural, 479  
Logical operators, 19  
Logical type, 17  
LOGOUT, 190  
LOOKUP(), 481  
Lotus files  
    exporting from SQL to, 660  
    importing to SQL from, 631  
Low-level file  
    writing to, 448  
Low-level file functions  
    FCLOSE(), 428  
    FCREATE(), 429  
    FEOF(), 431  
    FERROR(), 432  
    FFLUSH(), 433  
    FGETS(), 434  
    FOPEN(), 444  
    FPUTS(), 448

- FREAD(), 449
- FSEEK(), 451
- FWRITE(), 454
- Low-level file privileges, 429, 444
- LOWER(), 482
- Lowercase
  - checking for, 467
- LTRIM(), 483
- LUPDATE(), 483

## M

- Macro substitution, 21, 389
- Macro substitution in program loops, 389
- Macro terminator character, 389
- Macros in a DO WHILE loop, 143
- Master index
  - changing, 340
  - setting with ORDER, 340
- Math functions
  - FLOOR(), 443
  - LOG(), 479
  - MOD(), 493
  - PV(), 509
  - ROUND(), 518
  - SIGN(), 530
- Mathematical operators, 18
- MAX(), 484
- Maximum length
  - of FOR conditions, 164
  - of key expressions, 163
- Maximum number
  - of binary files, 185
  - of decimals, 312
  - of DO WHILEs, 143
  - of fields, 101
  - of GETs, 80
  - of locks, 680
  - of nested DO WHILEs, 143
  - of open files, 276
  - of parameters, 220
  - of parameters to a procedure, 139
  - of parameters to binary files, 185
  - of procedure files, 138
  - of procedures, 225, 365
  - of programmable keys, 438
  - of SORT fields, 263
  - of user-defined functions, 153
- Maximum size
  - of a record, 101
  - of an array, 119
  - of array names, 119
  - of character fields, 101
  - of numeric fields, 101
  - of the history buffer, 335
- Maximum width of message line, 353
- Maximum work areas, 10
- MCOL(), 485
- MDY(), 487
- MEMLINES(), 488
- Memo fields
  - effect of DBDEFINE, 611
  - extracting substrings, 516
  - greater than 64K, 62
  - overriding alignment in, 32
  - replacing part, 535
- memory allocation, 489
- Memory variable names, 266
- Memory variables. *See also* Memvar
  - as parameter, 220
  - defined, 14
  - system
    - streaming output, 25
- MEMORY(), 489
- Memvar
  - commands used to create, 14
  - created by FETCH, 622
  - entering row value to, 653
  - FETCH processing, 612, 622
  - in search conditions, 642
  - in SELECT clause, 640
  - INSERTing, 629
- Menu
  - deactivating, 114
  - determining name of, 490
- Menu commands
  - ACTIVATE MENU, 52
  - ACTIVATE POPUP, 53
  - CLEAR MENUS, 81
  - CLEAR POPUPS, 81
  - CLEAR SCREENS, 81

- CLEAR WINDOWS, 81
- DEACTIVATE MENU, 114
- DEACTIVATE POPUP, 115
- DEACTIVATE WINDOW, 116
- DEFINE BAR, 122
- DEFINE MENU, 125
- DEFINE PAD, 127
- DEFINE POPUP, 129
- DEFINE WINDOW, 130
- ON BAR, 193
- ON EXIT BAR, 199
- ON EXIT MENU, 201
- ON EXIT PAD, 203
- ON EXIT POPUP, 204
- ON MENU, 206
- ON PAD, 208
- ON PAGE, 209
- ON POPUP, 212
- ON SELECTION BAR, 214
- ON SELECTION MENU, 215
- ON SELECTION PAD, 216
- ON SELECTION POPUP, 217
- RELEASE MENUS, 236
- RELEASE POPUPS, 236
- RELEASE SCREENS, 236
- RELEASE WINDOWS, 236
- RESTORE SCREEN, 247
- RESTORE WINDOW, 247
- SAVE SCREEN, 254
- SAVE WINDOW, 255
- SHOW MENU, 260
- SHOW POPUP, 260
- Menu functions
  - BAR(), 398
  - BARCOUNT(), 399
  - BARPROMPT(), 400
  - MENU(), 490
  - PAD(), 499
  - PADPROMPT(), 500
  - POPUP(), 504
  - PROMPT(), 506
- Menu prompts, 127, 506
- MENU(), 490
- Message line, 353
- MESSAGE(), 491
- MIN(), 491

- MLINE(), 492
- MOD(), 493
- MODIFY COMMAND/FILE, 190
- Modulus function, 493
- MONTH(), 494
- Mouse commands
  - ON MOUSE, 207
  - SET MOUSE, 354
- Mouse functions
  - ISMOUSE(), 468
  - MCOL(), 485
  - MROW(), 495
- MOVE WINDOW, 192
- MROW(), 495
- MultiPlan files
  - exporting from SQL to, 660
  - importing to SQL from, 631
- Multiple index file, 163
  - effect of DBDEFINE, 610
- MVBLKSIZE, 179
- MVMAXBLKS, 179

## N

- Name of the active window, 550
- Names of procedures, 225
- Navigation keys, 146
- NDX(), 496
- Network functions
  - ACCESS(), 391
  - CHANGE(), 407
  - FLOCK(), 442
  - LKSYS(), 477
  - NETWORK(), 497
  - RLOCK(), 517
  - USER(), 547
- NETWORK(), 497
- Non-recursive commands, 156
- NOTE, 193
- Number of lines displayed, 319
- Number of programmable function
  - keys, 682
- Number of windows, 682
- Number of work areas, 682

- Numeric accuracy, 680
  - of type F numbers, 361
  - of type N numbers, 361
- Numeric comparison
  - MAX(), 484
  - MIN(), 491
- Numeric conversion
  - numeric expression to integer, 463
  - type F to type N, 437
  - type N to type F, 441
- Numeric display
  - in scientific notation, 17
- Numeric types, 16

## O

- Object code, 11
- Object file extension, 83
- Objects (SQL), 596, 667
- ON BAR, 193
- ON ERROR, 195–199
- ON ERROR exclusions, 196
- ON ESCAPE, 195–199
  - with READ, 197
- ON EXIT BAR, 199
- ON EXIT MENU, 201
- ON EXIT PAD, 203
- ON EXIT POPUP, 204
- ON KEY, 195–199
  - interference with GETs, 197
  - with SET TYPEAHEAD, 195
- ON KEY exclusions, 197
- ON MENU, 206
- ON MOUSE, 207
- ON PAD, 208
- ON PAGE, 209, 210
- ON POPUP, 212
- ON READERROR, 213
- ON READERROR exclusions, 214
- ON SELECTION BAR, 214
- ON SELECTION MENU, 215
- ON SELECTION PAD, 216
- ON SELECTION POPUP, 217
- OPEN (SQL), 594, 616, 617, 623, 633
- Opening CATALOG files, 291

- Opening database files, 275
- Opening procedure files, 365
- Operating system name, 499
- Operators, 15, 18
- ORDER BY Clause, 647, 649
  - in UNION query, 646
  - use of index, 598
- ORDER BY Clause (SQL), 646
- ORDER(), 498
- OS(), 499
- Output redirection, 14
- Outputting blocks of text, 269
- Overwrite prevention, 371

## P

- PACK, 219
- PAD(), 499
- PADPROMPT(), 500
- Page breaks, 209
  - in format files, 233
- Page handler command, 210
- Paragraph indenting, 557
- Parameter count, 502
- Parameter passing
  - to binary files, 185
- PARAMETERS, 219
- Parent file, 367
- Parentheses
  - to group SQL search conditions, 643
- Passing data to programs, 219
- Password file, 229
- Password protection (SQL), 625
- PATH in dBASE, 358
- Path specification
  - CREATE DATABASE, 596
- Pausing the display in SQL, 359
- PAYMENT(), 501
- PCOL(), 502
- PCOUNT(), 502
- PI(), 503
- PICTURE formatting of data, 541
- PLAY MACRO, 220
- Playing back macros, 221



- Pop-up menus
  - deactivating, 115
  - MESSAGE placement, 130
- POPUP(), 504
- Precedence
  - of border definitions, 288
  - of combined operators, 20
  - of field names over memvars, 139, 266
  - of fields over variables, 15
  - of format functions of @ , 43
  - of logical operators, 19
  - of mathematical operators, 19
  - of ON commands, 196
  - of ON KEY, 197
  - of operators, 19
  - of SET FIELDS, 290
- Predicates (SQL), 644
- Print form file, 26, 571
- Printed output control, 555
- Printer
  - control codes, 33
  - formfeeds, 559
  - linefeeds, 559
- Printer-formatted file, 363
- Printing
  - a box, 123
  - a formfeed with Postscript, 35
  - a left curly brace, 35
  - any character, 408
  - changed page numbers, 562
  - changing line spacing, 578
  - CHR(0), 33
  - condensed, 575
  - determining column, 502
  - determining row, 508
  - downloading Postscript file, 567
  - draft mode, 576
  - ejecting a blank sheet, 569
  - elite, 575
  - from a beginning page number, 562
  - from a program, 222
  - multiple copies, 565
  - on a network, 362
  - overstrike characters, 564
  - overstrikes, 32
  - page numbers, 561
  - pausing after each page, 579
  - pica, 575
  - positioning output, 564
  - quality mode, 576
  - querying for the print form file, 571
  - relative addressing, 508
  - selecting printer drivers, 566
  - sending ending control codes, 568
  - sending starting codes to the printer, 577
  - setting page offset, 574
  - setting the page length, 572
  - test labels, 174
  - testing printer status, 505
  - to a file, 363
  - to an ending page number, 570
  - to Postscript, 35
  - UDF controls on, 23
- PRINTJOB/ENDPRINTJOB, 222
- PRINTSTATUS(), 505
- PRIVATE, 224
- PRIVATE memory variables, 224
- Privileges (SQL), 667, 670, 671
  - adding, 627
  - ALTER, 592, 626
  - assigning, 624
  - checked before FETCH, 622
  - cumulative, 626
  - DELETE, 626
  - granting, 624
  - INDEX, 626
  - INSERT, 626, 629, 630
  - removing, 634
  - required for SELECT, 640
  - SELECT, 626
  - UPDATE, 626
- PRIVILEGES keyword (SQL), 626, 636
- PROCEDURE, 224
- Procedure file, 11
- Procedure limits, 13
  - size, 682
- Procedure list, 225
  - in object code files, 84
- Procedure names, 85, 225

Procedures, 11  
     search order, 12  
     system level, 12  
 Production .mdx file, 163  
 PROGRAM(), 506  
 Programmable function keys, 332  
 Programming commands in macros, 389  
 Programming function keys, 332  
 PROMPT FIELD, 129  
 PROMPT FILES, 129  
 PROMPT STRUCTURE, 129  
 PROMPT(), 506  
 Prompting for user input  
     with INPUT, 168  
     with WAIT, 279  
 Prompting user entry, 51  
 PROTECT, 227  
     in SQL, 610, 624, 626, 670  
 PROW(), 507  
 PUBLIC, 231  
 PUBLIC keyword (SQL), 626, 627, 636  
 PUBLIC memory variables, 231  
 PV(), 508

## Q

Qualifying group rows, 644  
 Query file extensions, 108  
 QUIT, 232  
 Quotes  
     in INSERT, 628  
     in SELECT search, 643

## R

Radians, 424  
 RAND(), 509  
 Random number generator, 509  
 RapidFile files  
     exporting from SQL to, 660  
     importing to SQL from, 631  
 RAT(), 510  
 READ, 233  
 READ and format files, 233  
 READKEY(), 511

READKEY() codes, 511  
 RECALL, 234  
 RECCOUNT(), 512  
 RECNO(), 155, 513  
 Recompilation  
     of object code files, 83  
 Record pointer  
     and filters, 328  
     and SET DELETED, 314  
     in LOCATE, 189  
     with SKIP, 261  
 record pointer  
     determining position, 155  
     restoring, 155  
 Records  
     determining number, 513  
     marked for deletion, 313, 417  
 RECSIZE(), 514  
 Recursive BROWSE restriction, 310  
 Redirecting  
     @...SAY output, 317  
     output to file, 284  
     output with SET PRINTER, 361  
     streaming output, 25  
 REINDEX, 235, 611  
 REINDEX keyword  
     and BLANK command, 69  
     in APPEND, 58  
 REINDEXing UNIQUE indexes, 235  
 Relating database files, 367  
 Relational operators, 18  
 Relative addressing with PCOL(), 502  
 Relative screen addressing, 411  
 RELEASE, 236  
 Releasing binary modules, 236  
 Removing blanks, 483  
 Removing memo fields  
     of deleted records, 219  
 Removing the integrity tag, 244  
 RENAME, 238  
 Repeating a character expression, 515  
 REPLACE, 239  
 REPLACE FROM ARRAY, 241  
 Replacing data, 239  
 REPLICATE(), 515  
 REPORT FORM, 242

RESET, 244  
 RESTORE, 245  
 RESTORE MACROS, 246  
 RESTORE SCREEN, 247  
 RESTORE WINDOW, 247  
 Restoring macros from a file, 246  
 Restoring memory variables, 245  
 Restoring windows from a file, 247  
 RESUME, 248  
 RESUME and SUSPEND, 248  
 RETRY, 248  
 RETURN, 249  
 Returning a result from a UDF, 250  
 Returning control to calling programs, 249  
 REVOKE (SQL), 625, 634, 670, 671  
 Right margin of output with ?/??, 580  
 RIGHT(), 516  
 RLOCK(), 516  
 ROLLBACK, 250, 617, 637, 638  
 ROLLBACK(), 517  
 Rolling dBASE out of memory, 522  
 ROUND(), 518  
 Rounding numbers, 518  
 Row (SQL)  
     cursor processing, 612, 633  
     deleting, 615  
     inserting, 629  
     ordering, 647  
     qualifying, 642, 644  
     summarizing information, 644  
 ROW(), 519  
 RTBLKSIZE, 179  
 RTMAXBLKS, 179  
 RTOD(), 519  
 RTRIM(), 520  
 RUN, 251  
 RUN(), 521  
 Running DGEN from dBASE, 419  
 RUNSTATS (SQL), 638, 669, 671

## S

Sample files, 683  
 SAVE, 252  
 SAVE MACROS, 253

SAVE SCREEN, 254  
 SAVE WINDOW, 255  
 Saving an image of the screen, 254  
 Saving keyboard macros to a file, 254  
 Saving memory variables to a file, 252  
 SCAN, 256  
 Scope keywords, 14  
 Screen areas for color settings, 300  
 Screen cursor position, 411  
 Screen refresh interval, 366  
 Search conditions (SQL)  
     defining, 642  
     in HAVING, 644, 646  
 Search order for procedures, 224  
 Searching  
     and record pointer position, 86  
     for names that sound alike, 420  
     indexed databases, 523  
     specific fields, 481  
     with LOCATE, 357  
     with LOCATE and CONTINUE, 86  
 Security, 227  
 Security considerations, 392  
 SEEK, 257  
     with SET DELETED, 258  
     with SET FILTER, 258  
     with SET NEAR, 258  
 SEEK and the record pointer, 258  
 SEEK expressions, 257  
 SEEK(), 523  
 SELECT, 258  
 SELECT (SQL), 594, 668  
     combining queries, 646  
     cursor definition, 613  
     cursor update, 613  
     cursor updates, 647  
     DECLARE CURSOR form, 652  
     defining a very, 639  
     defining search condition, 642  
     display-only form, 648  
     executing cursor, 633  
     grouping rows, 644  
     identifying tables, 641  
     in CREATE VIEW, 605  
     in HAVING, 644, 646  
     in INSERT, 629

- in UPDATE, 662
- into memvars, 640
- joining tables, 651
- ordering result, 644, 646
- qualifying rows, 642
- querying a view, 606
- querying Sysidxs, 599
- querying Syssyns, 601
- saving a result, 647
- SELECTing columns, 639, 640
- subquery, 651
- usage modes, 647, 648
- SELECT Clause (SQL), 640
- SELECT privilege (SQL), 626, 659
- SELECT(), 525
- SELECTing a work area by a variable, 259
- Self-join (SQL), 641
- Semicolon (;)
  - as SQL command terminator, 587
  - as wrap character, 244
- SET
  - disk, 283
  - display, 283
  - files, 283
  - in Config.db, 283
  - keys, 283
  - options, 283
- SET ALTERNATE, 284
  - exclusion of full screen operations, 284
- SET AUTOSAVE affected commands, 286
- SET BELL, 286
  - default duration, 286
  - default frequency, 286
  - frequency range, 287
- SET BLOCKSIZE, 287
- SET BORDER, 288
  - required order, 288
- SET CARRY, 290
- SET CATALOG, 291
  - adding entries, 292
  - and the ? query clause, 292
- SET CENTURY, 294
- SET CLOCK, 295
- SET COLOR, 296
  - standard area groups, 300
- SET commands
  - determining status, 526
- SET CONFIRM, 304
- SET CONSOLE, 305
- SET CURRENCY, 305
  - with FUNCTION clauses, 305
  - with PICTURE clauses, 305
- SET CURRENCY LEFT/RIGHT, 306
- SET CURSOR, 307
  - with the RUN command, 307
- SET DATE, 308
- SET DBTRAP, 309
  - and UDFs, 309
- SET DEBUG, 311
- SET DECIMALS, 311
- SET DEFAULT, 312
- SET DELETED, 313
- SET DELIMITERS, 314
  - and @...GET, 315
  - reset to default, 315
- SET DESIGN, 316
- SET DEVELOPMENT, 316
  - external editors, 317
- SET DEVICE, 317
- SET DIRECTORY, 318
- SET DISPLAY, 319
- SET ECHO, 320
- SET ENCRYPTION, 321
- SET ESCAPE, 322
- SET EXACT, 322
  - program compilation, 323
  - string comparison, 323
- SET EXCLUSIVE, 324
- SET FIELDS, 324
- SET FIELDS list respecting commands, 325
- SET FILTER, 328
  - work areas, 328
- SET FORMAT, 329
- SET FULLPATH, 331
- SET FUNCTION, 332
  - default function key assignments, 332
- SET HEADINGS, 333
  - column titles, 333
  - column width, 333
- SET HELP, 334
- SET HISTORY, 335

SET HOURS, 336  
 SET IBLOCK, 337  
 SET INDEX, 339  
 SET INSTRUCT, 341  
 SET INTENSITY, 342  
 SET KEY, 343  
 SET LDCHECK, 344  
 SET LIBRARY, 345  
 SET LOCK, 346  
 SET MARGIN, 349  
 SET MARK, 350  
 SET MBLOCK, 351  
 SET MEMOWIDTH, 352  
 SET menu, 283  
 SET MESSAGE, 353  
 SET MOUSE, 354  
 SET NEAR, 354  
     record pointer, 354  
 SET ODOMETER, 356  
 SET ORDER, 356  
     and SET ORDER TO TAG, 357  
 SET PATH, 358  
 SET PAUSE, 359, 641  
 SET POINT, 360  
 SET PRECISION, 360  
 SET PRINTER, 361  
 SET PROCEDURE, 364  
 SET REFRESH, 366  
 SET RELATION, 367  
 SET REPROCESS, 371  
 SET SAFETY, 371  
 SET SCOREBOARD, 372  
 SET SEPARATOR, 373  
 SET SKIP, 374  
 SET SPACE, 376  
 SET SQL, 376  
 SET STATUS, 377  
 SET STEP, 378  
 SET TALK, 378  
 SET TITLE, 379  
 SET TRAP, 380  
 SET TYPEAHEAD, 381  
 SET UNIQUE, 381  
 SET VIEW, 383  
 SET WINDOW, 384  
 SET(), 526  
 SET() keywords, 526  
 Setting the screen to black and white, 297  
 SHOW DATABASE (SQL), 656  
 SHOW MENU, 260  
 SHOW POPUP, 260  
 SIGN(), 530  
 Simple subquery, 651  
 SIN(), 531  
 Single index, 610  
 Single-database application, 657  
 SKIP, 261  
 SORT, 262  
 SORT compared to indexing, 263  
 SOUNDEX(), 531  
 Source code, 11  
 SPACE(), 533  
 Specifications of dBASE IV, 679  
 Specifying field delimiters, 314  
 Specifying the working directory, 318  
 SQL prompt, 7  
 SQL Prompt Interface, 7  
 Sqlcnt status variable (SQL), 623  
 Sqlcnt system variable (SQL), 633  
 Sqlcode status variable (SQL), 623  
 SQLDBA user ID (SQL), 597  
     catalog privileges, 668  
     drop privilege, 597, 599, 600, 606  
     REVOKE privilege, 636  
 SQRT(), 533  
 Square root function, 533  
 START DATABASE (SQL), 594, 657  
 Startup directory path, 456  
 Statistics (SQL), 639, 668  
 Status bar display, 377  
 Stepping through programs, 378  
 STOP DATABASE (SQL), 594, 619, 658  
 STORE, 265  
 STORE command  
     with elements and variables, 265  
 Storing averages to an array, 63  
 STR(), 534  
 String comparison, 322  
 String manipulation, 535  
 String operators, 19  
 STUFF(), 535

- Subquery
  - correlated, 649
  - simple, 651
- SUBSTR(), 536
- Substring comparison operator, 18
- Substring manipulation, 473
- Substrings
  - in SUBSTR(), 536
  - LEFT(), 473
  - RIGHT(), 516
- SUM, 267
- Suppressing clock display, 296
- Suppressing screen output, 305
- Suppressing the Organize menu, 55
  - in EDIT, 145
  - with INSERT, 169
- SUSPEND, 268
- Synonym (SQL)
  - catalog entry, 668, 674
  - creating, 601
  - deleting, 620
  - effect of DROP TABLE, 620
  - effect of DROP VIEW, 621
  - in INSERT, 627
  - in SELECT clause, 640, 641
  - naming, 601
  - use, 601
- Syntax (SQL), 587
- Syntax conventions, 13
- Sysauth catalog (SQL), 667, 670
- Syscolau catalog (SQL), 667, 671
- Syscols catalog (SQL), 667, 671
- Sysdbs catalog (SQL), 596, 600, 672
- Sysidxs catalog (SQL), 600, 667, 673
- Syskeys catalog (SQL), 667, 673
- SYSPROC, 346
- Syssyns catalog (SQL), 667, 674
- Systabls catalog (SQL), 668, 674
- System data format files
  - exporting from SQL to, 660
  - importing to SQL from, 631
- System date, 415
- System memory variables
  - defined, 24
  - relationships, 25

- System parameters
  - SET(), 526
- Systimes catalog (SQL), 668, 675
- Sysvdeps catalog (SQL), 668, 675
- Sysviews catalog (SQL), 668, 675

## T

- Tab stops, 582
- Table (SQL)
  - adding columns, 592
  - appending data, 630
  - assigning privileges to, 624
  - catalog entry, 667
  - creating, 602
  - deleting, 620
  - exporting data, 660
  - inserting rows, 627
  - naming, 602
  - synonym for, 600
- TABLE keyword (SQL), 626, 636
- Tables (SQL)
  - alias name for, 640
  - querying, 639
  - updating statistics about, 638
- TAG names, 163
- TAG(), 537
- TAGCOUNT(), 537
- TAGNO(), 538
- TAN(), 540
- Temporary indexes with NOSAVE, 357
- Testing
  - display type, 466
  - for alpha character, 464
  - for available work area, 525
  - for blank, 464
  - for color setting, 297
  - for lowercase, 467
  - for number of records, 512
  - for UNIQUE index, 544
  - for uppercase, 469
  - number of lines in memo, 488
  - user keypress, 472
- TEXT, 269
- Text editor, 190

TOTAL, 270  
Transaction log file, 64  
Transaction status, 412  
Transactions  
    commands not allowed in, 65  
    nesting, 65  
    NOLOG option in, 66  
    testing for data integrity, 467  
    testing outcomes, 65, 517  
Transactions (SQL)  
    removing SQL changes, 637  
    SQL commands disallowed in, 637  
TRANSFORM(), 541  
Trapping ESC key  
    after a READ, 197  
Trapping specific keys, 197  
Trapping the return key, 198  
Trigonometric functions  
    ACOS(), 393  
    ASIN(), 395  
    ATAN(), 397  
    ATN2(), 398  
    COS(), 413  
    DTOR(), 424  
    PI(), 504  
    RTOD(), 520  
    SIN(), 531  
    TAN(), 540  
TRIM(), 542. *See also* RTRIM()  
TYPE, 272  
TYPE(), 543  
Type-ahead buffer and error trapping, 381  
Type-ahead buffer size, 381

## U

UDF  
    defined, 21, 153  
    example of, 22  
    limitations, 23  
    names, 153  
    PARAMETERS in, 153  
    recursive commands in, 24  
    SQL limits on, 156  
UDF name limits, 153

Undoing transactions, 250  
Unencrypting files  
    with SET ENCRYPTION, 230  
UNION clause (SQL), 649  
UNIQUE index option, 164  
UNIQUE indexes, 381  
UNIQUE(), 544  
UNLOAD DATA (SQL), 611, 659  
UNLOCK, 273  
updatable view, 607  
    creating, 607  
    restrictions, 607  
UPDATE, 274  
UPDATE (SQL), 639, 647, 651, 661, 662  
    embedded form, 653  
    WHERE clause, 662  
    WHERE CURRENT OF clause, 662  
UPDATE privilege (SQL), 626  
Updating catalog files, 291  
UPPER(), 545  
USE, 275  
USE AGAIN, 278  
USE NOLOG, 277  
USE NOSAVE, 277  
User ID (SQL), returning, 640  
User keypress testing, 459  
USER keyword (SQL), 640  
USER(), 547  
User-defined functions. *See* UDF

## V

VAL(), 547  
Variables defined, 14  
VARREAD(), 548  
Version number of dBASE IV, 549  
VERSION(), 549  
View (SQL)  
    alias name for, 641  
    assigning privileges to, 624  
    catalog entry, 667, 674, 675  
    creating, 606  
    deleting, 621  
    effect of DROP TABLE, 621  
    inserting via, 627

- naming, 606
- naming columns, 606
- querying, 606, 640
- select use, 606
- SELECTing from, 640
- synonym for, 600
- updatable, 608, 627

View, dBASE

- effect of DBDEFINE, 611

View defined, 384

Viewing the fields list, 325

Virtual memory, 489

Virtual Memory Manager (VMM), 489

VisiCalc files

- exporting from SQL to, 660
- importing to SQL from, 631

## W

WAIT, 279

- capturing the keypress with SET ESCAPE, 279

WAIT precedence over ON KEY, 197

WHERE clause (SQL)

- in UPDATE, 662
- qualifying rows, 642
- specifying a join, 649

WHERE CURRENT OF clause (SQL), 647, 662

- in DELETE, 613, 615

Wildcard characters

- in SET FIELDS, 325

Wildcard comparisons, 475

Wildcards in filenames, 15

WINDOW color settings, 303

Window for memo fields, 384

WINDOW(), 550

WITH CHECK OPTION (SQL), 608

WITH GRANT OPTION (SQL), 626

Word wrapping

- the ??? output, 583

WORK keyword (SQL), 637

## Y

YEAR(), 550

## Z

ZAP, 280