



Language Reference



Visual dBASE®

dBASE Inc. Vestal, NY Santa Cruz, CA
<http://www.dbase2000.com> <news://news.dbase2000.com>

dBASE Inc. or Borland International may have patents and/or pending patent applications covering subject matter in this document. The furnishing of this document does not give you any license to these patents.

COPYRIGHT © 1984,1995 Borland International, 1999 dBASE Inc. All rights reserved. All dBASE product names are trademarks or registered trademarks of dBASE Inc. 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 the U.S.A.

Contents

Introduction	1	Object operators	22
How this book is organized	1	NEW operator.	22
Typographical conventions	2	Index operator	22
Chapter 1		Dot operator.	23
Language definition	3	Function operators.	23
Language elements	3	Call operator	23
Commands	3	Member call operator	24
Functions.	4	Index call operator	24
Built-in functions.	4	Scope resolution operator	25
User-defined functions (UDFs)	4	Precedence of operators.	26
System memory variables.	4	Chapter 2	
Preprocessor directives	5	Syntax conventions	27
Classes	5	Syntax notation	27
Standard classes	6	Guidelines to interpreting the syntax	28
Custom classes	6	Chapter 3	
dBASE statements	6	Language elements by category	31
Syntax	7	Commands and functions	39
Arguments.	7	Chapter 4	
Names	8	Commands and functions	41
Table and file names.	8	!.	41
Aliases	9	&&.	41
Database names	9	*	42
Field names	10	?	43
Memory variables	10	??	46
Arrays	11	???	46
Name skeletons.	11	@...CLEAR	47
Scope options.	11	@...FILL	47
Expressions	12	@...SAY...GET	47
Data types	12	@...SCROLL	47
Character data.	13	@...TO	48
Date data.	14	ABS()	48
Numeric data	14	ACCEPT	48
Logical data	14	ACCESS()	49
Memo data.	15	ACOPY().	49
Binary and OLE data.	15	ACOS().	50
Bookmark data	15	ACTIVATE MENU	51
NULL data type.	16	ACTIVATE POPUP.	51
Function-pointer data	17	ACTIVATE SCREEN.	52
Codeblock data	17	ACTIVATE WINDOW.	52
Statement codeblock.	18	ADEL().	52
Expression codeblocks	18	ADIR().	55
Object-reference data.	19		
Operators.	19		
Assignment operators	20		
Numeric operators	20		
Relational operators	20		
Logical operators	21		
String operators.	22		

AELEMENT().	57	CHANGE.	121
AFIELDS().	59	CHANGE().	122
AFILL().	60	CHARSET().	123
AGROW().	61	CHOOSEPRINTER().	123
AINS().	64	CHR().	124
ALEN().	67	CLASS...ENDCLASS.	125
ALIAS().	68	CLEAR	128
ANSI().	69	CLEAR AUTOMEM	129
APPEND.	70	CLEAR FIELDS	131
APPEND AUTOMEM.	72	CLEAR GETS	131
APPEND FROM	73	CLEAR MEMORY	131
APPEND FROM ARRAY	75	CLEAR MENUS.	132
APPEND MEMO.	77	CLEAR POPUPS	133
ARESIZE().	78	CLEAR PROGRAM	133
ASC().	81	CLEAR SCREENS.	134
ASCAN().	82	CLEAR TYPEAHEAD.	134
ASIN().	84	CLEAR WINDOWS	134
ASORT().	84	CLOSE...	135
ASUBSCRIPT().	87	CMONTH().	137
AT().	89	COL().	137
ATAN().	90	COMMIT().	138
ATN2().	91	COMPILE	139
AVERAGE.	92	CONTINUE	140
BAR().	93	CONVERT	141
BARCOUNT().	93	COPY	143
BARPROMPT().	94	COPY BINARY	145
BEGINTRANS().	94	COPY FILE.	146
BINTYPE().	96	COPY INDEXES.	147
BITAND().	97	COPY MEMO	148
BITLSHIFT().	98	COPY STRUCTURE	150
BITOR().	99	COPY TABLE	151
BITRSHIFT().	99	COPY TAG.	152
BITSET().	100	COPY TO ARRAY	153
BITXOR().	101	COPY TO...STRUCTURE EXTENDED	155
BLANK.	102	COS().	157
BOF().	103	COUNT.	159
BOOKMARK().	104	CREATE	160
BROWSE.	105	CREATE APPLICATION	161
BUILD	110	CREATE CATALOG.	162
CALCULATE	111	CREATE COMMAND.	164
CANCEL.	113	CREATE FILE	165
CATALOG().	114	CREATE FORM.	165
CD.	115	CREATE LABEL	167
CDOW().	116	CREATE MENU	168
CEILING().	117	CREATE POPUP	168
CENTER().	118	CREATE QUERY	169
CERROR().	120	CREATE REPORT	169

CREATE SCREEN	171	DOW()	224
CREATE SESSION	171	DTOC()	225
CREATE VIEW	174	DTOR()	226
CREATE VIEW...FROM ENVIRONMENT	174	DTOS()	227
CREATE...FROM	175	EDIT	228
CREATE...STRUCTURE EXTENDED.	177	EJECT	233
CTOD()	179	EJECT PAGE.	234
DATABASE()	180	ELAPSED()	235
DATE()	181	EMPTY()	237
DAY()	181	EOF()	238
DBERROR()	182	ERASE	239
DBF()	183	ERROR()	240
DBMESSAGE()	184	EXP()	241
DEACTIVATE MENU.	184	EXTERN	242
DEACTIVATE POPUP	185	FACCESSDATE()	247
DEACTIVATE WINDOW	185	FCLOSE()	247
DEBUG.	185	FCREATE()	248
DECLARE	187	FCREATEDATE()	250
DEFINE.	189	FCREATETIME()	250
DEFINE BAR	193	FDATE()	251
DEFINE BOX	193	FDECIMAL()	252
DEFINE COLOR	194	FEOF()	253
DEFINE MENU.	195	FERROR()	254
DEFINE PAD	195	FFLUSH()	255
DEFINE POPUP	196	FGETS()	256
DEFINE WINDOW	196	FIELD()	258
DELETE	196	FILE()	259
DELETE FILE	198	FIND	259
DELETE TABLE	198	FKLABEL()	261
DELETE TAG	199	FKMAX()	262
DELETED()	200	FLDCOUNT()	262
DESCENDING()	201	FLDLIST()	263
DIFFERENCE()	202	FLENGTH()	264
DIR/DIRECTORY	203	FLOCK()	265
DISKSPACE()	205	FLOOR()	267
DISPLAY.	206	FLUSH	268
DISPLAY COVERAGE	208	FNAMEMAX()	269
DISPLAY FILES.	209	FOPEN()	269
DISPLAY MEMORY.	211	FOR()	271
DISPLAY STATUS	212	FOR...NEXT	272
DISPLAY STRUCTURE	214	FOUND()	274
DMY()	216	FPUTS()	275
DO.	217	FREAD()	277
DO CASE.	219	FSEEK()	278
DO WHILE	220	FSHORTNAME()	279
DO...UNTIL	222	FSIZE()	280
DOS.	223	FTIME()	281

FUNCTION	282	LISTCOUNT().	335
FUNIQUE().	283	LISTSELECTED().	337
FV().	284	LKSYS().	338
FWRITE().	286	LOAD DLL.	340
GENERATE	288	LOCAL	341
GETCOLOR().	288	LOCATE	342
GETDIRECTORY().	289	LOCK().	344
GETENV().	290	LOG().	344
GETEXPR().	291	LOG10().	345
GETFILE().	292	LOGOUT.	346
GETFONT().	294	LOOKUP().	346
GO.	294	LOWER().	348
HELP	296	LTRIM().	349
HOME().	297	LUPDATE().	350
HTOI().	297	MAX().	350
ID().	298	MCOL().	351
IF.	299	MD.	352
IIF().	301	MDOWN().	352
IMPORT	302	MDX().	352
INDEX	303	MDY().	353
INKEY().	306	MEMLINES().	354
INPUT	307	MEMORY().	356
INSERT.	309	MENU().	356
INSERT AUTOMEM.	310	MESSAGE().	357
INSPECT().	312	MIN().	357
INT().	313	MKDIR	359
ISALPHA().	314	MLINE().	360
ISBLANK().	316	MOD().	360
ISCOLOR().	317	MODIFY...	361
ISLOWER().	317	MODIFY STRUCTURE	362
ISMOUSE().	318	MONTH().	364
ISTABLE().	319	MOVE WINDOW.	365
ISUPPER().	320	MROW().	365
ITOH().	320	MSGBOX().	365
JOIN.	321	NDX().	368
KEY().	323	NETWORK().	369
KEYBOARD.	324	NEXTKEY().	369
KEYMATCH().	324	ON BAR	371
LABEL FORM.	325	ON ERROR.	371
LASTKEY().	327	ON ESCAPE	372
LDRIIVER().	327	ON EXIT BAR	374
LEFT().	328	ON EXIT MENU	374
LEN().	330	ON EXIT PAD.	374
LENNUM().	331	ON EXIT POPUP	374
LIKE().	331	ON KEY	375
LINENO().	333	ON MENU.	377
LIST.	334	ON MOUSE	378

ON NETERORR	378	REFRESH.	428
ON PAD	379	REINDEX.	429
ON PAGE	379	RELATION()	430
ON POPUP	381	RELEASE.	431
ON READERROR	381	RELEASE AUTOMEM.	433
ON SELECTION BAR	382	RELEASE DLL.	434
ON SELECTION FORM.	383	RELEASE MENUS	435
ON SELECTION MENU	384	RELEASE OBJECT	435
ON SELECTION PAD	384	RELEASE POPUPS	436
ON SELECTION POPUP	384	RELEASE SCREENS	436
OPEN DATABASE.	385	RELEASE WINDOWS	437
OPEN FORM	386	RENAME.	437
ORDER().	387	RENAME TABLE.	438
OS().	388	REPLACE	439
PACK.	389	REPLACE AUTOMEM	441
PAD()	390	REPLACE BINARY.	443
PADPROMPT()	390	REPLACE FROM ARRAY.	445
PARAMETERS	391	REPLACE MEMO	447
PAYMENT()	391	REPLACE MEMO...FROM	448
PCOL().	393	REPLACE OLE	449
PCOUNT()	394	REPLICATE().	450
PI()	394	REPORT FORM	451
PLAY SOUND	395	RESOURCE().	452
POPUP().	397	RESTORE.	453
PRINTJOB...ENDPRINTJOB	397	RESTORE IMAGE	454
PRINTSTATUS().	398	RESTORE SCREEN.	456
PRIVATE.	399	RESTORE WINDOW.	456
PROCEDURE	401	RESUME	456
PROGRAM()	406	RETRY	457
PROMPT().	407	RETURN	458
PROPER().	408	RIGHT().	460
PROTECT	409	RLOCK().	461
PROW().	411	ROLLBACK().	463
PUBLIC.	412	ROUND().	464
PUTFILE().	414	ROW().	465
PV().	415	RTOD().	465
QUIT	417	RTRIM().	466
RANDOM().	417	RUN.	467
RAT().	419	RUN().	467
READ.	421	SAVE	469
READKEY().	421	SAVE SCREEN	470
READMODAL().	421	SAVE WINDOW	470
RECALL	423	SCAN	470
RECCOUNT().	424	SECONDS().	472
RECNO().	425	SEEK	473
RECSIZE().	426	SEEK().	475
REDEFINE.	427	SELECT.	476

SELECT()	477	SET INTENSITY.	526
SET	478	SET KEY	526
SET ALTERNATE	479	SET KEY TO	527
SET AUTOSAVE	481	SET LDCHECK	529
SET BELL.	482	SET LDCONVERT	530
SET BLOCKSIZE	483	SET LIBRARY	531
SET BORDER	484	SET LOCK	532
SET CARRY	484	SET MARGIN	534
SET CATALOG	485	SET MARK.	535
SET CENTURY	487	SET MBLOCK	536
SET COLOR OF.	488	SET MEMOWIDTH	538
SET COLOR TO.	488	SET MESSAGE	539
SET CONFIRM	489	SET MOUSE	539
SET CONSOLE	489	SET NEAR	539
SET COVERAGE	490	SET ODOMETER	540
SET CUAENTER	492	SET ORDER	541
SET CURRENCY	493	SET PATH	542
SET CURSOR	494	SET PCOL	543
SET DATABASE	495	SET POINT.	545
SET DATE	496	SET PRECISION.	546
SET DATE TO.	497	SET PRINTER	547
SET DBTYPE.	499	SET PROCEDURE	549
SET DECIMALS	500	SET PROW	551
SET DEFAULT	501	SET REFRESH.	552
SET DELETED	501	SET RELATION.	553
SET DELIMITERS	502	SET REPROCESS	557
SET DESIGN.	502	SET SAFETY	558
SET DEVELOPMENT	503	SET SEPARATOR.	559
SET DEVICE.	504	SET SKIP	560
SET DIRECTORY.	504	SET SPACE.	561
SET DISPLAY	506	SET STEP.	562
SET ECHO.	506	SET TALK	562
SET EDITOR.	506	SET TIME.	563
SET ENCRYPTION.	508	SET TITLE	564
SET ERROR	509	SET TOPIC.	565
SET ESCAPE.	510	SET TYPEAHEAD	566
SET EXACT	511	SET UNIQUE	566
SET EXCLUSIVE	513	SET VIEW	567
SET FIELDS	514	SET WINDOW OF MEMO	568
SET FILTER	516	SET()	569
SET FORMAT.	518	SETTO()	571
SET FULLPATH	518	SHELL()	573
SET FUNCTION	519	SHOW MENU.	574
SET HEADINGS	520	SHOW OBJECT	575
SET HELP	521	SHOW POPUP	576
SET IBLOCK.	522	SIGN()	576
SET INDEX	524	SIN()	577

SKIP	578
SLEEP	580
SORT	581
SOUNDEX()	584
SPACE()	586
SQLERROR()	587
SQLEXEC()	588
SQLMESSAGE()	589
SQRT()	590
STATIC	591
STORE	593
STORE AUTOMEM	595
STORE MEMO	596
STR()	598
STUFF()	599
SUBSTR()	601
SUM	602
SUSPEND	603
TAG()	605
TAGCOUNT()	606
TAGNO()	607
TAN()	608
TARGET()	609
TEXT	610
TIME()	610
TOTAL	611
TRANSFORM()	612
TRIM()	613
TYPE	614
TYPE()	615
UNIQUE()	617
UNLOCK	618
UPDATE	619
UPDATED()	620
UPPER()	620
USE	621
USER()	625
VAL()	625
VALIDDRIVE()	626
VARREAD()	627
VERSION()	627
WAIT	628
WINDOW()	629
WORKAREA()	629
YEAR()	630
ZAP	630

System memory variables 633

Chapter 5 System memory variables 635

_alignment	635
_app	636
_box	638
_curobj	639
_dbwinhome.	640
_indent	640
_lmargin	642
_padvance	643
_pageno.	644
_pbpage.	645
_pcolno	646
_pcopies.	647
_pdriver.	648
_pject.	649
_pepage.	650
_pform	651
_plength	653
_plineno.	654
_ploffset.	656
_porientation.	656
_ppitch	657
_pquality	658
_pspacing.	659
_rmargin	661
_tabs.	662
_wrap	663

Preprocessor directives 665

Chapter 6 Preprocessor directives 667

#define	667
#if	670
#ifdef	671
#ifndef.	672
#include.	674
#pragma	675
#undef.	676

Classes 677

Chapter 7

Classes 679

CLASS ARRAY	679
CLASS ASSOCARRAY	681
CLASS BROWSE	682
CLASS CHECKBOX	686
CLASS COMBOBOX	690
CLASS DDELINK	693
CLASS DDETOPIC	694
CLASS EDITOR	697
CLASS ENTRYFIELD	700
CLASS FORM	704
CLASS IMAGE	708
CLASS LINE	710
CLASS LISTBOX	712
CLASS MENU	716
CLASS MENUBAR	719
CLASS OBJECT	721
CLASS OLE	721
CLASS OLEAUTOCLIENT	725
CLASS PAINTBOX	726
CLASS POPUP	729
CLASS PUSHBUTTON	731
CLASS RADIOBUTTON	735
CLASS RECTANGLE	739
CLASS SCROLLBAR	742
CLASS SHAPE	745
CLASS SPINBOX	746
CLASS TABBOX	749
CLASS TEXT	752

Properties 755

Chapter 8

Properties 757

AbandonRecord()	757
ActiveControl	757
Add()	759
Advise()	760
Alias	760
Alignment	761
Anchor	763
Append	763
AutoSize	764

Before	765
BeginAppend()	766
Border	770
BorderStyle	771
Bottom	772
CanClose	773
Checked	774
ClassName	775
Close()	776
ColorHighlight	777
ColorNormal	777
Copy()	779
Count()	782
CUATab	783
CurSel	784
Cut()	785
DataLink	785
DataSource	786
Default	788
Delete	789
Delete()	790
DesignView	791
Dimensions	792
Dir()	793
DirExt()	794
DisabledBitmap	795
DoVerb()	796
DownBitmap	798
DropDownHeight	799
EditCopyMenu	800
EditCutMenu	801
EditPasteMenu	801
EditUndoMenu	802
Element()	803
Enabled	804
EscExit	805
Execute()	806
Fields	807
Fields()	808
FieldWidth	809
Fill()	810
First	811
FirstIndex	812
FocusBitmap	813
Follow	814
FontBold	815
FontItalic	815

FontName	816	OnClick	861
FontSize	817	OnClose	862
FontStrikeOut	818	OnExecute	863
FontUnderline	819	OnFormSize	864
Function	820	OnGotFocus	865
GetTextExtent()	821	OnHelp	866
Group	822	OnInitMenu	867
Grow()	823	OnKeyDown	868
Header3D	824	OnKeyUp	869
Height	826	OnLeftDblClick	869
HelpFile	827	OnLeftMouseDown	871
HelpID	828	OnLeftMouseUp	873
hWnd	828	OnLostFocus	875
Icon	829	OnMiddleDblClick	876
ID	830	OnMiddleMouseDown	878
Initiate()	831	OnMiddleMouseUp	880
Insert()	832	OnMouseMove	882
IsIndex()	833	OnMove	884
IsRecordChanged()	833	OnNavigate	885
Key	834	OnNewValue	886
Keyboard()	835	OnOpen	887
Left	836	OnPaint	888
LineNo	837	OnPeek	889
LinkFileName	837	OnPoke	890
Maximize	839	OnRightDblClick	891
MaxLength	840	OnRightMouseDown	893
MDI	841	OnRightMouseUp	895
MenuFile	842	OnSelChange	897
Minimize	843	OnSelection	898
Mode	844	OnSize	899
Modify	845	OnUnadvise	901
MousePointer	846	Open()	901
Move()	847	PageCount()	902
Moveable	848	PageNo	903
Multiple	849	Parent	905
Name	850	Paste()	906
NextCol()	851	PatternStyle	906
NextIndex()	852	Peek()	907
NextObj	852	Pen	908
NextRow()	853	PenStyle	909
Notify()	854	PenWidth	910
OldStyle	855	Picture	911
OleType	855	Poke()	912
OnAdvise	856	PopupMenu	913
OnAppend	857	Print()	914
OnChange	858	RangeMax	915
OnChar	859	RangeMin	916

RangeRequired	917	TrackRight	958
ReadModal()	918	Unadvise()	958
Reconnect()	919	Undo()	959
Refresh()	919	UpBitmap	960
Release()	921	Valid.	961
RemoveAll()	922	ValidErrorMsg.	962
RemoveKey()	922	ValidRequired	963
Resize()	923	Value	964
Right	924	Vertical	965
SaveRecord()	925	View.	965
ScaleFontName	925	Visible.	967
ScaleFontSize	926	When	968
Scan()	927	Width	969
ScrollBar	929	WindowMenu	970
SelectAll	930	WindowState	971
Selected()	931	Wrap	972
Separator	931		
Server	932	SQL	975
ServerName	933		
SetFocus()	934	Chapter 9	
ShapeStyle	935	Local SQL	977
ShortCut	936	Memory variable substitution in SQL queries.	977
ShowDeleted	936	Naming conventions	977
ShowHeading	937	Table names	977
ShowRecNo	938	Column names	978
ShowSpeedTip	939	ALTER TABLE	978
Size	940	CREATE INDEX	979
Sizeable.	940	CREATE TABLE	979
Sort()	941	DELETE FROM	981
Sorted.	942	DROP INDEX	982
SpeedBar	943	DROP TABLE	982
SpeedTip	944	INSERT INTO	983
SpinOnly	945	SELECT	984
StatusMessage.	945	UPDATE	985
Step	946		
Style.	947	Appendixes	987
Subscript()	948		
SysMenu	949	Appendix A	
TabStop.	950	Changes since dBASE IV 2.0	989
Terminate()	951	New language elements	989
Text	952	Enhanced language elements	994
TimeOut	953	Unsupported language elements.	998
Toggle.	953		
Top	954		
Topic	955		
TopMost	956		

Appendix B		
Visual dBASE specifications	1001	
.DBF tables	1001	
Indexes	1001	
Fields	1002	
Multiuser.	1002	
Procedures.	1002	
Files	1003	
Miscellaneous	1003	
Appendix C		
File structures	1005	
Table header and records	1005	
Table header structure	1005	
Table records	1006	
Binary, memo, and OLE fields and .DBT files	1007	
Appendix D		
INKEY() and READKEY() values	1009	
Appendix E		
ASCII character chart		
(code page 437)		1013
		1014
Appendix F		
Error codes		1015
Index		1031

The *Language Reference* describes the dBASE language as it's implemented in *Visual dBASE*. It contains specification-level documentation for the commands, functions, system memory variables, preprocessor directives, and classes.

Use this manual to look up how a particular command or function works, what syntax is required, and for examples on how to use it. You can obtain the same information in online Help.

How this book is organized

- Chapter 1, “Language definition,” describes the basic concepts—components of the dBASE language, statement elements, data types, expressions, and operators.
- Chapter 2, “Syntax conventions,” describes the symbols and conventions used in presenting the syntax of language elements, and provides guidelines for interpreting the syntax notation.
- Chapter 3, “Language elements by category,” lists all the commands, functions, and other language elements categorically by the type of operation they perform.
- Chapter 4, “Commands and functions,” provides an alphabetical listing and description of commands and functions.
- Chapter 5, “System memory variables,” provides an alphabetical listing and description of the system memory variables available to control various printer and environmental settings.
- Chapter 6, “Preprocessor Directives,” provides an alphabetical listing and description of the preprocessor directives which allow you to control program compilation.
- Chapter 7, “Classes,” provides an alphabetical listing and description of the standard classes that *Visual dBASE* provides.
- Chapter 8, “Properties,” provides an alphabetical listing and description of the standard class properties.

- Chapter 9, “Local SQL,” describes the syntax of SQL commands that can be used within dBASE when working with non-database server data (i.e. dBASE and Paradox tables).
- Appendix A, “Changes since dBASE IV,” summarizes the changes to the dBASE language since dBASE IV; it lists the new, changed, and discontinued commands and functions.
- Appendix B, “Visual dBASE specifications,” provides specifications of *Visual* dBASE capacities and limits.
- Appendix C, “File structures,” describes the structure of *Visual* dBASE table and memo files.
- Appendix D, “INKEY() and READKEY() values,” lists the values returned by the INKEY() and READKEY() functions.
- Appendix E, “ASCII character chart,” lists the character codes in code page 437.
- Appendix F, “Error codes,” lists *Visual* dBASE error codes and messages.

Typographical conventions

The *Language Reference* uses specific typographical conventions to help you distinguish among the various language and syntax elements. These conventions are used to make the manual more readable.

Convention	Applies to	Examples
Italic	Variable names, array names, arguments	The <i>cNames</i> variable, array <i>aCosts</i> , <filename> argument
ALL CAPS	Command and function names and keywords, DOS files and directories	BROWSE, EOF(), INDEX ...TAG CUSTOMER.DBF
Initial caps	Procedures, applications, table names, field names, menu commands.	Accounts application, Price field
Camel caps	Property names	OnLeftDbIClk
Monospaced font	Code examples	USE Names

In addition to the typographical conventions listed in this table, Chapter 2 provides specific information for interpreting the symbols used in the syntax of commands, functions, and other language elements.

Language definition

The dBASE language is a structured programming language designed primarily for developing database applications. It consists of a collection of standard commands, functions, and classes that you use to create applications to store, manage, and process data.

This chapter defines the standard language elements in dBASE and describes the basic items needed to construct *statements* that you can enter in the Command window or in a program. A statement is a complete instruction that directs dBASE to perform a specific task.

Language elements

The standard language elements in *Visual* dBASE are

- Commands
- Functions
- System memory variables
- Preprocessor directives
- Classes

Commands

Commands tell dBASE to perform certain actions. A statement often starts with a command (statements are sometimes called *command statements*). Although a statement can contain another language element, such as a function, it can contain only one command.

The following statement, for example, uses the ? command and the DATE() function to display the current date:

```
? DATE( )
```

Most commands consist of options (sometimes called *clauses*) that you can use to tailor the command's operation to accomplish specific tasks. With the USE command, for example, you can simply open a table, or add the EXCLUSIVE option to open the table exclusively.

```
USE mytable           && opens mytable.dbf
USE mytable EXCLUSIVE && opens mytable.dbf exclusively
```

A statement can begin with either a command keyword or with one of several different *implicit* commands. Implicit commands include

- A memory variable assignment, which is an implicit STORE command:

```
x=45    && same as STORE 45 TO x
```

- A record number, which is an implicit GO command:

```
23      && same as GO 23
```

Functions

dBASE recognizes two kinds of functions: built-in and user-defined. A function returns a value and consists of a keyword followed by left and right parentheses. The parentheses can contain arguments that are passed to the function for processing. In the following statement, for example, the SQRT() function returns the square root of 36:

```
? SQRT(36)
```

Built-in functions

dBASE built-in functions generally perform arithmetic, text, date, logical, or conversion operations. Some functions, such as RLOCK() and SEEK(), perform an action and return a value. Some functions are useful for querying purposes; RECNO(), for example, returns the number of the current record, and DBF() returns the name of the table in the current work area.

User-defined functions (UDFs)

UDFs are functions you create and call in a program just as you would call a built-in function. To define a UDF, use the FUNCTION command followed by the name of the UDF, an optional parameter list, and the sequence of commands to execute when the UDF is called. For more information about creating UDFs, see the FUNCTION command in Chapter 4 of this manual and Chapter 4 in the *Programmer's Guide*.

System memory variables

System memory variables are memory variables that dBASE maintains automatically. These variables are typically used to specify environmental and printer settings, and to control the format of printed and screen output.

At start-up, dBASE initializes system memory variables to their defaults. The defaults for some system memory variables that control printer settings, such as `_pdriver` and `_porientation`, are determined by the Windows printer setup defaults.

To distinguish them from user-defined memory variables, system memory variables begin with the underscore (`_`) character.

Like memory variables that you define, system memory variables follow the same scoping rules. When a program that contains privately declared system memory variables has finished running, the variables automatically revert to their original settings.

Unlike user-defined memory variables, however, system memory variables cannot be released from memory. You can only change and query their values.

Preprocessor directives

Preprocessor directives are statements you insert in your dBASE programs to instruct the compiler how to compile the program. You can use them to replace text in a program, set up conditional compilation, or specify compiler options.

With these capabilities, you can, for example, maintain one code base and compile different portions of your code depending on the platform it runs on.

Preprocessor directives begin with a number sign (`#`). For more information about using preprocessor directives, see Chapter 7 in the *Programmer's Guide*.

Classes

Classes are specifications, or templates, for creating objects. *Visual* dBASE provides many standard classes that you can use to create common Windows objects, such as radio buttons, pushbuttons, and entry fields.

When you create an object, you create an *instance* of that object's class. The object will have the predefined attributes specified in the class. These attributes are called *properties*. Once you create an object, you can customize its characteristics by assigning values to the properties. Some object properties have code associated with them, and perform operations on the object. These properties are called *methods*.

As a programmer, you work with objects as collections of memory variables, where each property and method is a memory variable. This makes it easy for you to dynamically add properties to and change the properties of an object.

The following example shows how to create an entry field from the `Entryfield` class, and shows some of the properties you can set:

```
DEFINE ENTRYFIELD CustomerName OF CustomerForm;  
PROPERTY;  
    Width 20.00;;  
    Height 2.00;;  
    Top 7;;  
    Left 18;;  
    FontSize 10.00;;
```

```
DataLink "Customer->CustName"  
...
```

As the example illustrates, it would be a simple matter to change the size of the entry field—just assign different values to the Width and Height properties.

Standard classes

Most of the standard classes that *Visual* dBASE provides are specifications for standard Windows objects (also called *controls*) that you create and place on user-defined windows called forms. To create your own objects, you need to create the corresponding classes first (see the next section).

Custom classes

You can create new classes from scratch, or derive them from the standard classes. To do so, use the CLASS...ENDCLASS command. A class that is derived from another class is called a *subclass*; it inherits the properties and methods of the class it is based on. You can then customize the attributes of the subclass by adding, deleting, or changing properties.

The following example shows how to create a new class, Quitbutton, based on the standard Pushbutton class:

```
CLASS Quitbutton(f) OF PUSHBUTTON(f)  
    this.text = "Quit"  
    this.FontName = "Arial"  
    this.FontBold = .T.  
    this.FontSize = 12  
    this.OnClick = {;QUIT}  
    ...  
ENDCLASS
```

Once you've defined a new class, you can create objects from it. The following example creates a button from the Quitbutton class and places it on the CustomerForm form:

```
DEFINE QUITBUTTON Quit OF CustomerForm;  
PROPERTY;  
    Width 10.00,;  
    Top 18.00,;  
    Height 2.00,;  
    ...
```

For more information about classes and objects, and programming with them, see chapters 9 through 12 in the *Programmer's Guide*.

dBASE statements

You write statements to tell dBASE the operations you want to perform. Statements have a general structure—most begin with a command keyword and usually contain options and arguments.

Syntax

The structure of a statement is called its *syntax*. The following example shows the syntax for the REPLACE command:

```
REPLACE <field 1> WITH<exp 1> [ADDITIVE]
    [, <field 2> with <exp 2> [ADDITIVE]...]
    [ALL | REST | NEXT <expN> | RECORD <expN>]
    [FOR <condition 1>]
    [WHILE <condition 2>]
    [REINDEX]
```

The following table identifies the syntax elements of the REPLACE command:

Syntax element	Items in the REPLACE command
Command keyword	REPLACE
Options	ADDITIVE, ALL, REST, NEXT, RECORD, FOR, WHILE, REINDEX
Arguments	<field 1>, <field 2>, <exp 1>, <exp 2>, <expN>, <condition 1>, <condition 2>

This manual uses special typographical conventions in command syntax to indicate the rules for constructing a statement. Chapter 2 describes the syntax conventions in detail. The following table shows some of the key syntax symbols:

Symbol	Description
[]	Indicates an optional part of the syntax.
< >	Indicates an argument that you must supply; for example, <expN> indicates that you must enter a numeric expression.
	Indicates two or more mutually exclusive options for example, ON OFF.
...	Indicates an item that can be repeated any number of times; for example, [, <field 2> WITH <exp 2> [ADDITIVE]...] indicates that you can specify more than one field to replace.

Note dBASE allows you to abbreviate the names of commands, functions, and command options to the first four characters when creating a dBASE statement. You can, for example, abbreviate MODIFY COMMAND to MODI COMM. This, however, is not recommended; abbreviating statements decreases their readability, and you don't obtain any increase in performance (since statements are compiled before execution). Furthermore, because dBASE supports custom classes and properties, you can't abbreviate those names. The Pushbutton class, for example, can't be abbreviated to Push because dBASE will assume that Push is another class.

Arguments

An argument is a word or sequence of words enclosed in angle brackets (< >) that identifies a part of the syntax you need to supply, such as <scope>, <memvar>, <field>, <filename> or <exp>. When entering an argument in a statement, don't enter the angle brackets.

You can specify either a literal value or an expression in place of any argument. The following shows part of the REPLACE command syntax, then an example of the values or expressions (shown in bold) you can use with the command:

Syntax:

```
REPLACE <field 1> WITH<exp 1> [ADDITIVE]
    [, <field 2> with <exp 2> [ADDITIVE]...]
    [ALL | REST | NEXT <expN> | RECORD <expN>]
    [FOR <condition 1>]
    ...
```

Example of statement:

```
REPLACE ZipCode WITH "54261" FOR ZipCode = "54260"
```

The following sections describe some of the common types of arguments you use in statements.

Names

Some commands and functions require you to supply a name, such as a file name, table name, field name, memory variable, and so on. Names can be up to 32 characters in length and are made up of any combination of letters, digits, and the underscore (_) character with the following exceptions:

- DOS file names have an 8-character limit (with a file extension of up to 3 characters) and must follow DOS naming conventions.
- You can also specify a table, field, or memory variable name that includes spaces by delimiting the name with colon characters, for example, *:new name*.
- Variable and field names must start with an alphabetic character.

The following sections, “Table and file names,” “Aliases,” “Database names,” “Memory variables,” and “Field names” describe in more detail rules for specifying the different types of names.

Table and file names

Note Earlier versions of dBASE referred to .DBF files as database files. *Visual* dBASE refers to .DBF files as tables and uses the term database for a collection of tables. Database also refers to SQL databases that you access through the Borland SQL Link.

You can specify a file name as a literal or as a character expression that evaluates to a file name. You can also use the & macro substitution operator to specify the name of a file at run time.

In *Visual* dBASE (and also in dBASE IV), you should use the more efficient method of indirect reference wherever you are asked to provide a file name. For example:

```
Mfile = "Client"    && Name of existing table
USE (Mfile)
```

A full file name consists of the drive and path where a file resides on disk, the file’s root name, and the file’s extension, that is,

```
[<drive>] [<path>] <root name> [ <file extension> ]
```

For example, you could enter

```
C:\VISUALDB\SAMPLES\ANIMALS.DBF
```

The full directory and file name can be up to 79 characters. The drive, path, and file-name extension portion of the full file name are optional. All file-name restrictions follow DOS conventions.

When you know a file's location relative to the current directory, you can use the DOS current (.) and *parent* (..) directory notation to specify a file's location. For example,

```
USE ..\clients
```

You can use wildcard characters ? or * in file names to specify a file-name skeleton for referencing any file names that match a given pattern. In most cases, dBASE displays the Open File dialog box to let you choose the particular table you want to open.

Aliases

An alias is an alternative name for an open table or an open table's work area. When you access a table in *Visual* dBASE, a work area (in memory) is used to open the table and any associated files such as an index file. Only one work area can be current at a time.

You use the alias in a statement to identify a work area and to access a table open in that work area. You can identify the work area by number (1 through 225), by letter (A through J), or by alias name. *Visual* dBASE lets you open up to 225 tables at a time by allotting each open table its own unique work area.

When you open a table with *USE*, *Visual* dBASE automatically establishes the table's name as an alias for the table. (You can use the *ALIAS()* function to identify the alias for any open table.) If you prefer, you can specify your own alias, by providing a name of up to 32 characters, for example,

```
USE Clients ALIAS Contacts
```

Just as you can with file names, you can specify an alias as a literal or an expression. When a literal alias name might be confused with another name or command option, use a work area expression. The result of a work area expression identifies a work area containing the table you want to access. To avoid mistaking the name of a memory variable that contains a work area with a literal value, enclose the variable name in parentheses as shown in the following example:

```
cAliasName = "Contacts"  
SELECT (cAliasName)
```

Database names

Visual dBASE uses database names or aliases (assigned with BDE Configuration Utility BDECFG.EXE) to specify a drive and directory location from which to access tables and files that are all related to the same application. (Refer to Appendix B in *Getting Started* for more information on using the BDECFG.EXE utility.)

You can also define database names to specify the location from which to access tables on a database server. To access database servers, you need to install the Borland SQL Link.

Using *Visual dBASE*, you can have only one database active at a time (although multiple databases can be open at the same time). Like tables and work areas, you can specify a database name as a literal or an expression. For example, to use a memory variable to specify the name of a database to open a database, you could enter

```
cDatabaseName = "MktgSqlServer"  
OPEN DATABASE (cDatabaseName)
```

After opening databases, you can specify one of them as the current database using the SET DATABASE command.

Field names

A *field name* is the name of one *field*, or item of information, contained in a record of a table. For example, LASTNAME might be the name of a field that contains clients' last names. In each record in the table, a client's last name would be entered in the LASTNAME field.

When you specify a field name as an argument in a command or function (for example, TRIM(LASTNAME)), dBASE assumes you are referring to a field in a table in the current work area. If the field you want to access isn't in the table in the current work area, you need to qualify the field name with an alias. A qualified field name includes a reference to an alias (designating the work area) and the field name in that aliased table. To qualify a field name, use the alias symbol (->) between the alias name and the field name, as shown in the following example:

```
Client->Lastname
```

In the preceding example, the expression specifies the LASTNAME field in the Client table.

Note When fields and memory variables have the same name, fields take precedence over memory variables. To distinguish the memory variable, prefix it with m->, for example m->Lastname.

Memory variables

Memory variables are named locations in memory where you store data values: strings, numbers, dates, logical values, codeblocks, object references, or function pointers. You assign each of these values a name so that you can later retrieve them.

You can use these values to store user input, perform calculations, do comparisons, or define values that are used as arguments for other dBASE commands and functions. You can create a memory variable by assigning it a value using the = operator or the STORE command. The statements in the following example are equivalent:

```
cProduct = "Visual dBASE"  
STORE "Visual dBASE" TO cProduct
```

The scope of memory variables is usually limited to the dBASE program or procedure file in which it is declared or defined. To explicitly define the scope, you can use the

PUBLIC, PRIVATE, LOCAL, or STATIC commands. For more information, refer to the command descriptions in Chapter 4 of this manual and Chapter 5 in the *Programmer's Guide*.

Arrays

Visual dBASE also lets you create a special set of memory variables called an array. An array is an n -dimensional table of values stored in memory. Each entry in the array is called an element, and each element in an array can be treated like a memory variable and can also be included in an expression.

To create an array, you can use the DECLARE command, or you can create an array as an object. When you use arrays in an expression, follow the same rules as those for memory variables. You can use an array element in any expression that accepts a memory variable, and you can store the same types of data in an array element as you can in a memory variable.

Name skeletons

Name skeleton arguments let you specify wildcard characters ? and * as part of a character string to qualify field, file, or index names or to define a search string. For example, you could enter the following command to select all fields that begin with the *c* character:

```
SET FIELDS TO ALL LIKE c*
```

The * specifies that zero or more characters can appear in the position; the ? specifies that any single character can appear in the position within the string where the ? is located. A dBASE skeleton differs from a DOS skeleton in that the wildcard characters can appear anywhere and in any order, for example:

```
SET FIELDS TO ALL LIKE *name?
```

Scope options

In commands that process records in a table, you can use a scope keyword to specify the number of records to process. All commands that include a <scope> option have a default scope, usually ALL or NEXT 1. The following table describes the keywords you can use for the scope argument.

Keyword	Description
ALL	Processes all records in the table starting with the first.
REST	Starting with the current record, processes all subsequent records to the end of the table.
NEXT <expN>	Starting with the current record in the table, processes the next <expN> number of records.
RECORD <expN>	Processes only the record specified with <expN>.

In addition, many commands that process records allow you to specify FOR and WHILE conditions, which further qualify the records to be processed. Records are processed in natural order or by the order specified by a master index, if one is in use. Also, records are qualified by any filter condition or query file in use.

Expressions

You use expressions to provide values for arguments in a command statement. A simple expression might be a single data item, such as a field name, a constant, or a memory variable name. For example, the implicit STORE command requires an expression for its argument:

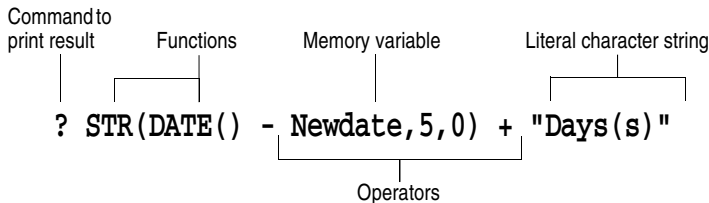
```
Name = "Peter"
```

In that example, "Peter" is a character expression.

More complex expressions are formed by combining several data items with operators. You can create expressions with one or more of the following elements:

- Field names
- Memory variables, including system memory variables and array elements
- Constants
- Functions
- Operators

The following illustration shows an example of a complex expression, STR(DATE() - Newdate, 5, 0) + " Day(s)", in a statement:



Operators (described in a later section) link each of these elements so that dBASE can evaluate the entire expression as a single unit.

When you combine fields, memory variables, constants, and a function's returned value in an expression, they must be the same data type (except for dates, which can be added to a number). If necessary, you can use functions to convert elements of differing data types to one common type, usually character. The example in the previous illustration uses the STR() function to convert a number to a character value before concatenating it with a character string.

The following sections describe all the *Visual* dBASE data types and operators that you can use to form expressions.

Data types

In dBASE, all data items are identified by type. The basic data items—fields, memory variables, constants, and functions—are assigned data types depending on how the item

is created. For example, the type of expression you assign to a memory variable determines the variable's data type, and a function's return value defines its data type.

Note To find out an item's data type, use the TYPE() function.

The following table lists the data types that *Visual* dBASE supports and the value that the TYPE() function returns for each:

Data type	TYPE() returns
Character	C
Date	D
Numeric	N
Float	F
Logical	L
Memo	M
Binary	B
OLE	G
Bookmark	BM
Function-pointer	FP
Codeblock	CB
Object-reference	O

The following sections describe each of the supported data types.

For information about working with the various data types, see Chapter 6 in the *Programmer's Guide*.

Character data

Character data consists of any combination of letters, digits, spaces, and special symbols. In .DBF tables, a character field can contain up to 254 characters; a character memory variable or expression can contain up to 32,766 characters.

Character constants or literal character strings must be enclosed in matching single or double quotation marks or brackets, as shown in the following examples:

```
a = 'text'
b = "text"
c = [text]
```

You can also store an ASCII decimal value to a character string with the CHR() function. The following statements both create a character memory variable containing the letter A:

```
STORE "A" TO cLetter
STORE CHR(65) TO cLetter
```

You can nest character strings in a literal character string by enclosing the nested string in a different pair of delimiters. The following example shows a statement and its result:

```
? "This is a 'character' string"
This is a 'character' string
```

Note A literal character string can't contain the null character (ASCII 0), the EOF character (ASCII 26), or the LF character (ASCII 10).

Date data

Date data provides a way to store a calendar date that is represented in a format such as *mm/dd/yy*. Internally, however, date data is always stored in the format *yyyymmdd*.

The size of a date variable or field is always eight bytes. *Visual* dBASE validates date information whenever it is entered or changed. By default, the format for dates is set by the International option of the Windows Control Panel. You can override this format with the SET DATE command or with the DATE setting in the DBASEWIN.INI file.

You specify a literal date value by enclosing it within braces ({}). You can also specify a date as a character string and convert it to a date type by using the CTOD() function. For example, {12/20/59} is the same as CTOD("12/20/59").

To specify a blank date, specify {}, { / / }, or CTOD(" / / ").

You can subtract a date from another date. The result is a number that represents the number of days between the dates. You can also add or subtract a number (representing a number of days) from a date. The result will also be a date.

Numeric data

The numeric data type lets you perform mathematical operations (additions, multiplications, and other mathematical functions) on data.

When designing the structure of a table, you can specify numeric or float types for numeric fields. You can specify up to 20 digits with up to 18 decimal places for both numeric and float types.

Unlike dBASE IV, however, there is no difference in the way *Visual* dBASE processes numeric or float data. In mathematical operations, *Visual* dBASE provides the same precision level (19 digits) for both types. In dBASE IV, numeric and float data are processed differently—numeric type numbers can have precision of up to 20 digits and float numbers have precision of 15. In *Visual* dBASE, the float type is maintained for backward compatibility with dBASE IV.

If you assign a number to a memory variable, the TYPE() function always returns N. The TYPE() function returns F for float fields only.

If you use very large or very small numbers, *Visual* dBASE displays numbers in exponential scientific notation, such as .6E + 23.

Logical data

Logical fields and memory variables store either a true (.T.) or false (.F.) value. Logical fields or variables accept T, t, Y, or y for true and F, f, N, or n for false. In statements or expressions, you must delimit logical values with periods (for example, STORE .T. to Mlogic).

Memo data

Memo data is any data stored in memo fields. A memo field is a variable-length field used to hold large blocks of characters (its size is limited only by available memory). Memo fields are typically used for text manipulation, although you can also store the contents of OLE fields, bitmaps, images, sound, or any other binary data in a memo field. (*Visual* dBASE adds two new data types to specifically store OLE and binary data respectively.)

dBASE stores the actual memo data in an associated memo file, which has a .DBT file-name extension.

Binary and OLE data

Visual dBASE provides two new data types, binary and OLE, to store binary and OLE data in table fields. It supports the standard binary type files including .BMP, .PCX, and .WAV files.

dBASE stores the actual binary or OLE data in the associated .DBT file.

Bookmark data

A bookmark is a reference to a specific record in a table. It is similar to the record pointer in dBASE.

dBASE supports bookmarks primarily for Paradox and SQL tables, which don't have record numbers. Because Paradox and SQL tables don't identify records by record number, you can't, for example, use GO 23 to go to record 23, or use RLOCK("1, 2, 3") to lock records 1, 2, and 3.

Bookmarks let you earmark records so that you can navigate to and perform operations on them. To mark the current record, use BOOKMARK() to store the bookmark value to a memory variable.

The following example shows how to use bookmarks to access records in a Paradox table:

```
USE Contact.db           && Open a Paradox table
bMark1= BOOKMARK()       && Mark the first record
SKIP 2                   && Advance 2 records
bMark2 = BOOKMARK()      && Mark the record
? RLOCK("bMark1, bMark2", 1) && Lock 1st and 3rd records in work area 1
GO bMark1                 && Go back to the first record
```

The bookmark value is a special unprintable data type. You can't query a bookmark; if you assign a bookmark value to a memory variable, then query the value of the variable, dBASE simply displays "BookMark." The following example illustrates this:

```
Record_a = BOOKMARK()
? Record_a    && displays "BookMark"
```

You can, however, compare bookmark values in a table. Using operators, such as >, =, or <, you can compare BOOKMARK() values to determine the relative positions of two records in a table. The following example illustrates this:

```
USE Contact.db
Record_a = BOOKMARK()
GO BOTTOM      && Go to the last record
Record_b = BOOKMARK()
? Record_a < Record_b  && Returns .T. because Record_a appears before Record_b
```

All commands and functions, such as RLOCK() and EDIT, that accept a record pointer argument also accept a bookmark.

NULL data type

Visual dBASE supports the concept of a null value. NULL is a constant representing the absence of a value, as opposed to a value of 0 for a numeric or an empty string for a character variable. NULL is not supported in fields of .DBF files, but may be supported by other database file formats which Visual dBASE can access.

When Visual dBASE is used to access tables which support null values, NULL can be used to set a field to a null value or to retrieve a null value from a field. Therefore, each of the following would be legal statements:

```
REPLACE TABLE->FIELD WITH NULL
SET FILTER TO TABLE->FIELD <> NULL
SEEK NULL
```

Null can be used as a value in an expression. In general, if the expression needs to manipulate the value passed to it, and that value is null, the expression will return NULL. For example:

```
? SIN(0)      && Returns 0.00
? SIN(NULL)   && Returns NULL
? UPPER("")   && Returns an empty character string
? UPPER(NULL) && Returns NULL
? 4 + 0       && Returns 4
? 4 + NULL    && Returns NULL
```

If a function does not need to manipulate the value passed, the NULL will be converted to its default value for the type of argument the function expects. For example:

```
? TIME()      && Returns the time to whole second
? TIME(0)     && Returns the time to hundredths of a second
? TIME(NULL)  && Returns the time to hundredths of a second, NULL treated as 0
```

Table 1.1 shows the internal default values for NULL that dBASE supports.

Table 1.1

Type	Default value
Bookmark	"" (empty character string)
Character	blank character(s)
Date	{/ /} (empty date)

Table 1.1

Type	Default value
Logical	.F.
Numeric	0
Unknown	0, "", .F., or {/}

The default data type of NULL is unknown. NULL can be assigned a data type by using it in an expression expecting a certain type, although it retains the value NULL:

```
x = NULL      && x assigned NULL of unknown type
? TYPE("x")   && Returns "U"
? UPPER(x)    && Returns NULL of character type, x still unknown type
x = SIN(NULL) && Returns NULL of numeric type and assigns to x
? TYPE("x")   && Returns "N", x is numeric but still has NULL as value
? UPPER(x)    && Data type mismatch error, UPPER() expects a character type parameter
```

Function-pointer data

As its name suggests, the function pointer is a reference to a function or procedure. Function pointers make it easy for you to define functions and procedures, then call them when needed.

All event properties, such as `OnClick` and `OnOpen`, in the standard objects are variables of either function pointer type or codeblock type (codeblocks are described in the next section). With a function pointer, you can assign a constant or variable reference that points to a function or procedure.

The following example shows how to specify a reference to the `Go_Next` procedure for the `OnClick` event property:

```
DEFINE PUSHBUTTON Nextbutton OF CustomerForm;
    Property;
    OnClick Go_Next
...

PROCEDURE Go_Next
    IF .NOT. EOF()
        SKIP
    ELSE
        GO TOP
    ENDIF
```

Codeblock data

A codeblock is a logical grouping of one or more dBASE statements, or a dBASE expression. Like the function pointer, it is a means of executing dBASE statements, and is particularly useful when used with an object's event property.

Unlike the function pointer, however, codeblocks are literal statements that you assign to the event property. A codeblock is a data type that *contains* dBASE code. Like other data types, you can assign a codeblock to a memory variable or pass it as a parameter.

Visual dBASE provides two types of codeblocks: *statement* and *expression*. Statement codeblocks can contain one or more dBASE statements, while expression codeblocks can contain only dBASE expressions.

Statement codeblock

The syntax and rules for writing a statement codeblock follow:

```
{[|<parameters>|]; <statement> [|; <statement> ...]}
```

- The braces, { }, are required.
- If you pass parameters, they must be delimited by | |.
- Precede each statement with a semicolon (;).

The following example shows how to assign a statement codeblock to the OnClick event property:

```
DEFINE PUSHBUTTON Quitbutton OF CustomerForm;  
    Property;  
    OnClick {;QUIT}  
    ...
```

The following example shows how to assign a statement codeblock with parameters to a memory variable:

```
AddNum = {|a,b|; ? a+b}
```

To execute the codeblock, use the following:

```
AddNum(10,10)    && Returns 20
```

Expression codeblocks

The syntax and rules for writing an expression codeblock follow:

```
{[|<parameters>|] <expression>}
```

- The braces, { }, are required.
- If you pass parameters, they must be delimited by | |.

The following example shows how to assign an expression codeblock to the Valid event property:

```
DEFINE ENTRYFIELD Salary OF EmployeeForm;  
    PROPERTY;  
    Valid {Salary >= 10000}  
    ...
```

The following example shows how to assign an expression codeblock with parameters to a memory variable; unlike the statement codeblock, a semicolon is not required after the parameter declaration:

```
AddNum = {|a,b| a+b}
```

To execute the codeblock, use the following:

```
AddNum(10,10)    && Returns 20
```


For information about techniques for using codeblocks, see Chapter 4 in the *Programmer's Guide*.

Object-reference data

The object-reference data type provides a way to reference *Visual* dBASE objects and the collection of variables (properties and methods) they contain.

When you create an object, *Visual* dBASE creates an *object reference variable* that refers to the object. Unlike other memory variables that actually contain a value, such as a character string, object reference variables contain a *reference* to the object, not the object itself.

When you assign one object reference variable to another, you don't duplicate the object; you simply have another reference to the same object.

The following example shows how you use the NEW operator to create a form object from the Form class (there are two ways to create objects; you can use the NEW operator or the DEFINE command):

```
CustomerForm = NEW Form( ) && CustomerForm is the object reference variable
```

Once you've created the form, you can set its properties by referencing the property variables. As discussed earlier, an object is a collection of variables, and each property is a variable.

The following example shows how you use the dot (.) operator (described in the next section on operators) to reference an object's property:

```
CustomerForm.Text = "Customer Data"  
CustomerForm.Width = 80.00  
CustomerForm.Top = 2.00  
CustomerForm.Left = 2.00  
CustomerForm.Height = 24.75
```

As the example illustrates, you reference an object's properties by specifying the object reference variable and the property variable.

For information about objects and object-oriented programming, see chapters 9 through 12 in the *Programmer's Guide*.

Operators

An operator is a symbol or keyword that performs an operation on data or expressions. *Visual* dBASE provides the following types of operators:

- Assignment
- Numeric
- Relational
- Logical
- String
- Object

- Function
- Scope resolution

Note dBASE provides another operator, & (the macro substitution operator), which is not described in this section. Refer to Chapter 5 in the *Programmer's Guide* for more information on its use.

All operators require either one or two arguments, called *operands*. Those that require a single operand are called *unary operators*; those requiring two operands are called *binary operators*.

The following is an example of a unary operator, .NOT.:

```
lTrue = .T.
? .NOT. lTrue && returns .F.
```

The following is an example of a binary operator, * :

```
? 7 * 7
```

Assignment operators

dBASE provides the equals (=) operator to assign values of expressions to memory variables. For example, you can use the equals operator to store a character string:

```
x = "Store this string in x"
```

The assignment operator can be used with expressions of any data type.

Numeric operators

Numeric operators perform calculations and generate numeric results. Here are the numeric operators that *Visual* dBASE provides:

Operator	Description
+	Addition/unary positive
-	Subtraction/unary negative
*	Multiplication
/	Division
** or ^	Exponentiation
()	Parentheses for grouping (to explicitly specify the order in which expressions are evaluated)

Numeric operators are used with numeric expressions. You can also use numeric operators with date expressions to subtract one date from another date, and to add or subtract a number from a date (where the number represents a number of days).

Relational operators

Relational operators are used to compare two expressions of the same data type. The comparison results in a logical true (.T.) or false (.F.) value. The operators can be used

with character, numeric, date, and logical expressions. Here are the relational operators that *Visual* dBASE supports.

Operator	Description
<	Less than
>	Greater than
=	Equal to
==	Exactly equal to
<> or #	Not equal to
<= or =<	Less than or equal to
>= or =>	Greater than or equal to
\$	Substring comparison

Character string comparisons are case-sensitive and are affected by the current language driver and the setting of the SET EXACT command. When SET EXACT is OFF, character strings match if all characters in the string on the right of the equal (=) sign match the beginning characters in the string on the left; the string on the right can be shorter, but not longer, than the string on the left.

When SET EXACT is ON, the characters in both strings must be identical. Trailing blanks, however, are ignored. The following example shows how the SET EXACT setting works:

```
SET EXACT OFF
? "abcde" = "abcd"    && Returns .T.
? "abc" = ""         && Returns .T.
? "abc" = "abcd"     && Returns .F.
SET EXACT ON
? "abcde" = "abcd"   && Returns .F.
? "abc" = "abc "    && Returns .T. (trailing blanks ignored)
```

To impose an exact comparison whether SET EXACT is ON or OFF, use == instead of =.

The \$ (contained in) string comparison operator compares two character strings to determine if the character string on the left is contained in the character string on the right. For example, A\$B returns true if character string A is either identical to character string B or contained within B. You can specify a substring as a literal character string, a character memory variable, a character or memo field.

Logical operators

Logical operators are used to compare two logical expressions and generate a logical true (.T.) or logical false (.F.) result. The operators *Visual* dBASE supports are the following:

Operator	Description
.AND.	Logical AND; returns true if both expressions are true.
.OR.	Logical OR; returns true if either of the expressions is true.

Operator	Description
.NOT.	Logical NOT; returns false if expression is true, true if expression is false.
()	Parentheses for grouping (to explicitly specify the order in which expressions are evaluated).

String operators

String operators concatenate two or more character strings into a single character string. *Visual dBASE* supports the following string operators:

Operator	Description
+	Concatenates two strings, leaving trailing spaces between the strings intact when the strings are joined
-	Concatenates two strings, moving trailing spaces in the first string to the end of the last string
()	Parentheses for grouping (to explicitly specify the order in which expressions are evaluated)

Object operators

Object operators are used to create and reference objects, properties, and methods. *Visual dBASE* supports the following object operators:

Operator	Description
NEW	Creates a new instance of an object
[]	Index operator, which accesses the contents of an object through a numeric value
. (period)	Dot operator, which accesses the contents of an object through an identifier name

NEW operator

The NEW operator creates an object of a specified class.

The following is the syntax for the NEW operator:

```
<object name> = NEW <class name>([<parameters>])
```

The following example shows how to use the NEW operator to create a form object from the Form class. By default, dBASE creates an object reference variable with the same name you give the form (CustomerForm in this example).

```
CustomerForm = NEW Form()
```

The object exists as long as there are references to it.

Note The DEFINE command is another way to create an object.

Index operator

The index operator, [], accesses an object's properties or methods through a value, which is usually a number. The following shows the syntax for using the index operator (often called the array index operator):

<object reference>[<expN>]

You typically use the index operator to reference elements of array objects, as shown in the following example:

```
TArray = NEW Array(20)    && Create a new array object with 20 elements
TArray[1] = 10            && Change the value of the 1st element to 10
? TArray[1]              && Returns 10
```

Dot operator

The dot operator, ("."), accesses an object's properties or methods through a name. The following shows the syntax for using the dot operator:

<object reference>.[<object reference>].<property name>

The dot operator is the most common way to refer to an object's properties or methods. The following statements illustrate how you use it to assign values:

```
DEFINE FORM CustFrm      && Create a new form object
CustFrm.Text = "Customers" && Set the Text property of CustFrm
CustFrm.OnOpen = OpenProc && Set the OnOpen event property of CustFrm
```

If an object contains another object, you can access the child object's properties by building a path of object references leading to the property, as the following statements illustrate:

```
DEFINE ENTRYFIELD NameField OF CustFrm && Create an entry field in the CustFrm form
CustFrm.NameField.Width = 20          && Set the Width property of NameField in CustFrm
```

Function operators

Function operators are used to call functions and procedures. *Visual* dBASE supports the following function operators:

Operator	Description
<fp>([params])	Call operator; calls a function or procedure associated with a function pointer <fp>.
<obj-ref>.<variable name>([params])	Member call operator; calls the function, procedure, or codeblock associated with a specified code-reference variable.
<obj-ref>[<exp>](<params>)	Index call operator; similar to member call operator but calls the function or codeblock with an index

Call operator

The call operator, (), calls a procedure or codeblock associated with a specified function-pointer or codeblock variable. It also passes any specified parameters to the procedure or codeblock. The following is the syntax for the call operator:

<function pointer|codeblock variable>([parameters])

The following statements are examples of how you use the call operator to call a procedure associated with a function pointer:

```
? Square(10)      && Calls procedure Square; returns 100
? TYPE("Square")  && Returns FP; Square is a constant of type function pointer
```

```

Calc = Square      && Creates a variable of type function pointer
? Calc(7)          && Calls procedure Square; returns 49

PROCEDURE Square
PARAMETERS n
RETURN n*n

```

The following statements show how you use the call operator to call a codeblock:

```

x = {|z| z + 1}    && Creates a variable of type codeblock
? x(6)            && Calls the codeblock stored in the variable

```

Member call operator

Like the call operator, the member call operator calls the procedure or codeblock associated with a specified function-pointer or codeblock variable. The difference is that the function-pointer or codeblock variable is contained within an object.

In addition, when you call the procedure or codeblock, dBASE creates a special object reference variable called *this*. The *this* variable refers to the object that contains the function-pointer or codeblock variable. The following is the syntax for the member call operator:

```
<object reference>.<function pointer|codeblock variable>([parameters])
```

The following example shows how to use the member call operator:

```

x = NEW OBJECT()      && Creates a new object
x.Proc = DisplayVal   && Assigns the procedure to a variable of function pointer type
x.Val = 7             && Assigns a number to the Val variable
? x.Proc()            && Calls the procedure in the fp variable in x; returns 7

PROCEDURE DisplayVal
RETURN this.val

```

As the example illustrates, the *this* variable (like the object reference variable *x*) references the object. In fact, in the procedure declaration, using `RETURN x.val` gives the same results as `RETURN this.val`. The difference is that using *x* hard codes the object reference; you would be able to return *val* only to object *x*. With *this*, you can reuse the procedure as-is in any other object.

Index call operator

The index call operator works the same way as the member call operator but calls the procedure or codeblock with an index value. As with the member call operator, dBASE creates a special object reference variable called *this* when the procedure or codeblock is called. The following is the syntax for the index call operator:

```
<function pointer|codeblock variable>[expN]([parameters])
```

You can access unnamed members of a class using the index, just as you would refer to array elements:

```

x = NEW ARRAY()      && Creates a new array object
x[1] = MyProc        && Assigns the procedure to array element 1
x.val = 10           && Assigns a number to the Val variable
? x[1]()             && Calls the procedure stored in element 1; returns 10

```

```
PROCEDURE MyProc
RETURN this.val
```

Scope resolution operator

The scope resolution operator (::) lets you reference methods in a class. This is useful if you've built a hierarchy of classes and you want to call a method from a superclass. You can, for example, modify an inherited method in a subclass by calling the method directly from the superclass, then adding to its behavior in the subclass. For more information about this subject, see Chapter 11 in the *Programmer's Guide*.

The following is the syntax for using the scope resolution operator:

```
<class name> | class | super :: <method name>
```

As the syntax indicates, you can refer to the superclass by its class name, or one of the keywords, *class* or *super*. The *class* keyword references the current class (the class being declared); *super* references the superclass of the current class.

The following example shows how the BeepMove pushbutton accesses the BeepProc procedure from its superclass, BeepButton, then adds new functionality to it:

```
CLASS BeepButton(form) OF MyButton(form)
    this.text = "Beep"
    this.Onclick = BeepProc

    PROCEDURE BeepProc
        ? CHR(7)
    RETURN
ENDCLASS

CLASS BeepMove(form) OF BeepButton(form)
    this.text = "Beep and Move"                && Overrides Text property of superclass

    PROCEDURE OnClick
        super::BeepProc()                    && References the BeepProc method of superclass
        this.Left = this.Left+1              && Adds functionality to the OnClick method
    RETURN
ENDCLASS
```

Note The `super::BeepProc()` statement is the same as `BeepButton::BeepProc()`.

With the scope resolution operator, you can also call a method from a class. As with any procedure, the class declaration must be in the same program file as the call statement, or be loaded as a procedure file with SET PROCEDURE. The following example illustrates how you use the scope resolution operator to call a method in a class:

```
? MyClass::SayHello()    && Calls the method in MyClass; prints "Hello"

CLASS MyClass
    PROCEDURE SayHello
        ? "Hello"
```

```
RETURN .T.  
ENDCLASS
```

Precedence of operators

You can combine several different operators in a single expression. The way in which *Visual* dBASE will evaluate an expression is determined by the order in which different operators are evaluated. The following table lists the precedence in which *Visual* dBASE evaluates operators.

Operator	Description
()	Parentheses grouping, all expressions
[]	Field name, character memory variable, and array indexing
->	Alias symbol
.(period)	Dot operator
NEW	New object operator
+, -	Unary plus (+) and minus (-) signs
**, ^	Exponentiation
*, /	Multiplication and division
+, -	Addition and subtraction
=, <>, <, <=, >, >=, \$	Relational operators
.NOT., .AND, .OR.	Logical operators

You can use parentheses to group expressions and explicitly specify the order in which you want expressions evaluated. If you nest parentheses, expressions in the innermost parentheses are evaluated first.

Syntax conventions

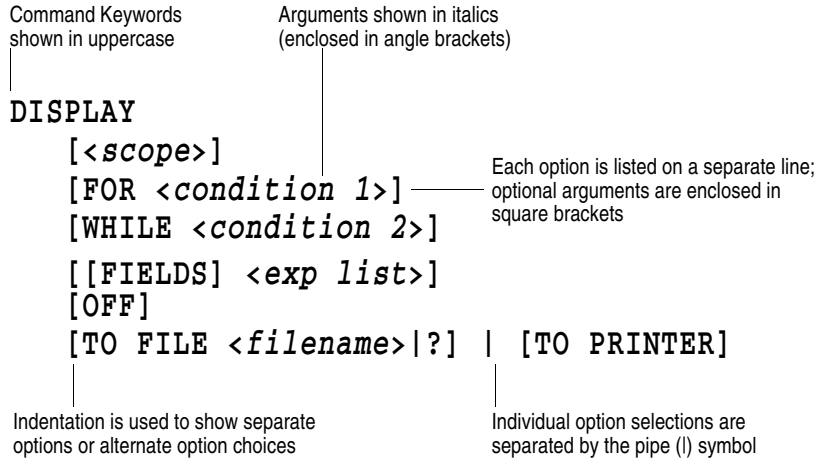
The *Language Reference* uses specific symbols and conventions in presenting the syntax of dBASE language elements. This chapter describes the symbols used in syntax and provides information on interpreting the syntax conventions.

Syntax notation

The following table describes the symbols used in syntax:

Symbol	Description
[]	Indicates an optional item
< >	Indicates an argument that you must supply; for example, <expN> indicates that you must enter a numeric expression
{ }	Indicates a codeblock
	Indicates a codeblock parameter list
	Indicates two or more mutually exclusive options for example, ON OFF
...	Indicates an item that may be repeated any number of times; for example, [, <field 2> WITH <exp 2> [ADDITIVE]...] indicates that you can specify more than one field to replace
,	List item separator
;	In a codeblock, separates each command statement In a command line, indicates that the command statement continues on the next line

The following illustration shows the syntax for a typical command.



Unlike dBASE command and function keywords, which are shown in uppercase letters, property names are capitalized differently. Property names are camel-capped, that is, they contain both uppercase and lowercase letters if the name consists of more than one word. If the property is a method, the name is followed by parentheses. Examples of properties include OnAppend, OnRightMouseDown, Checked, and Close().

These conventions help you differentiate the language elements; for example,

- DELETE is a command
- Delete is a property
- DELETED() is a function
- Delete() is a method

These typographical conventions are for readability only. When writing code, you can use any combination of uppercase and lowercase letters.

Note You can abbreviate most keywords in the dBASE language to the first four characters; however, for clarity, this is not recommended. You must refer to classes and properties by their full name, since you can define your own custom classes and properties, and *Visual* dBASE needs the full name to identify the correct one to use.

Guidelines to interpreting the syntax

This section tells you how to interpret some of the more complex syntax conventions.

- Options are shown in alphabetical order except in cases where it is logical or required to provide them in another order, for example:

```
DEFINE <class name> <object name>
  [OF <container object>]
  [FROM <row, col> ...]
```

When creating a dBASE statement, you can usually include the options in any order.

- The pipe symbol (|) separates mutually exclusive options. If there are more options that can fit on a single line, additional options appear indented following the pipe symbol ending the first line, as shown in the following example:

```
MONO | COLOR | EGA25 | EGA43 | MONO43 |  
      VGA25 | VGA43 | VGA50
```

- If a keyword precedes one of many alternate option keywords, each alternate keyword is shown on its own line indented from the keyword with which it is used:

```
PROMPT  
  ARRAY <array> |  
  FIELD <field> |  
  FILES [LIKE <filename skeleton>] |  
  STRUCTURE [IN <alias expC>]
```

In this example, you could select any of four options, PROMPT ARRAY..., PROMPT FIELD..., PROMPT FILES..., or PROMPT STRUCTURE...

- The bracket, [], indicates that the item is optional; sometimes, optional items appear nested in other optional items, as shown in the following example:

```
ACOPY(<source array name>, <target array name>  
      [, <starting element expN> [, <elements expN> [, <target element expN>]]])
```

With ACOPY(), you specify a list of arguments, each delimited with a comma. The third, fourth, and fifth arguments are optional. Notice that the third and fourth arguments do not have right brackets immediately after the argument. This means that in order to specify a value for the fourth argument, you must specify a value for the third argument. Similarly, in order to specify a value for the fifth argument, you must specify values for the previous two arguments. Contrast this syntax notation with the next one.

- Sometimes, optional items appear nested in other optional items, but are independent of each other, as shown in the following example:

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

Like the previous ACOPY() example, SET COLOR TO takes a list of optional arguments, each delimited with a comma. Notice, however, that each argument is contained within its own pair of square brackets. This means that you can specify any one of those arguments independently of the others. If you omit an argument from the list, mark its position in the list with a comma. For example, SET COLOR TO , , B means that the perimeter is set to the color blue.

- Arguments for which you substitute a particular value or expression are shown in italics enclosed in angle brackets. In many cases, instead of just specifying the type of data required (for example, <expC> or <expN>), the syntax provides a more descriptive argument name, such as:

```
<target element expN>
```


Language elements by category

dBASE contains over 600 commands, functions, standard classes, system memory variables, and preprocessor directives. If you're not familiar with the dBASE language, this chapter will help you learn what language elements are available to perform programming tasks. It groups commands and functions into categories based on their purpose.

Note The category name appears to the right of each language element listed in this book.

Programs

&&	CREATE COMMAND	IF	QUIT
*	DO	IIF()	RETURN
BUILD	DO CASE	MODIFY COMMAND	SCAN
CANCEL	DO WHILE	NOTE	SET DEVELOPMENT
CLEAR PROGRAM	DO...UNTIL	PARAMETERS	SET LIBRARY
CLOSE PROCEDURE	FOR...NEXT	PCOUNT()	SET PROCEDURE
COMPILE	FUNCTION	PROCEDURE	SLEEP

Memory variables

ACOPY()	AGROW()	ASUBSCRIPT()	RELEASE
ADEL()	AINS()	CLEAR MEMORY	RESTORE
ADIR()	ALEN()	DECLARE	SAVE
AELEMENT()	ARESIZE()	LOCAL	STATIC
AFIELDS()	ASCAN()	PRIVATE	STORE
Afill()	ASORT()	PUBLIC	

Error handling and debugging

CERROR()	GENERATE	PROGRAM()	SET STEP
DBERROR()	LINENO()	RESUME	SQLERROR()
DBMESSAGE()	LIST COVERAGE	RETRY	SQLMESSAGE()
DEBUG	MESSAGE()	SET COVERAGE	SUSPEND
DISPLAY COVERAGE	ON ERROR	SET ECHO	
ERROR()	ON READERROR	SET ERROR	

String Data

ANSI()	LEFT()	RAT()	STUFF()
AT()	LEN()	REPLICATE()	SUBSTR()
CENTER()	LIKE()	RIGHT()	TRANSFORM()
DIFFERENCE()	LOWER()	RTRIM()	TRIM()
ISALPHA()	LTRIM()	SET EXACT	UPPER()
ISLOWER()	OEM()	SOUNDEX()	
ISUPPER()	PROPER()	SPACE()	

Numeric data

ABS()	EXP()	PAYMENT()	SET POINT
ACOS()	FLOOR()	PI()	SET PRECISION
ASIN()	FV()	PV()	SET SEPARATOR
ATAN()	INT()	RANDOM()	SIGN()
ATN2()	LENNUM()	ROUND()	SIN()
CEILING()	LOG()	RTOD()	SQRT()
COS()	LOG10()	SET CURRENCY	TAN()
DTOR()	MOD()	SET DECIMALS	

Date and time data

CDOW()	DOW()	SECONDS()	SET MARK
CMONTH()	ELAPSED()	SET CENTURY	SET TIME
DATE()	MDY()	SET DATE	TIME()
DAY()	MONTH()	SET DATE TO	YEAR()
DMY()			

Expressions and type conversions

ASC()	DTOS()	ITOH()	STR()
CHR()	EMPTY()	MAX()	TYPE()
CTOD()	GETEXPR()	MIN()	VAL()
DTOC()	HTOI()		

Table basics

ALIAS()	COPYTO...STRUCTURE EXTENDED	IMPORT	SELECT()
APPEND FROM	CREATE	ISTABLE()	SET CATALOG
CATALOG()	CREATE CATALOG	LIST STRUCTURE	SET DATABASE
CLOSE ALL	CREATE...FROM	MODIFY CATALOG	SET DBTYPE
CLOSE DATABASES	CREATE...STRUCTURE EXTENDED	MODIFY STRUCTURE	SET TITLE
CLOSE TABLES	DATABASE()	OPEN DATABASE	SQLEXEC()
COPY	DBF()	REFRESH	USE
COPY STRUCTURE	DELETE TABLE	RENAME TABLE	WORKAREA()
COPY TABLE	DISPLAY STRUCTURE	SELECT	

Fields and records

APPEND	COUNT	LUPDATE()	REPLACE OLE
APPEND AUTOMEM	DELETE	MEMLINES()	SET AUTOSAVE
APPEND FROM ARRAY	DELETED()	MLINE()	SET BLOCKSIZE
APPEND MEMO	EDIT	PACK	SET CARRY
BINTYPE()	EOF()	RECALL	SET DELETED
BLANK	FDECIMAL()	RECCOUNT()	SET FIELDS
BOF()	FIELD()	RECNO()	SET MBLOCK
BOOKMARK()	FLDCOUNT()	RECSIZE()	SET MEMOWIDTH
BROWSE	FLDLIST()	RELEASE AUTOMEM	SET WINDOW OF MEMO
CHANGE	FLENGTH()	REPLACE	SKIP
CLEAR AUTOMEM	FLUSH	REPLACE AUTOMEM	STORE AUTOMEM
CLEAR FIELDS	GO	REPLACE BINARY	STORE MEMO
COPY BINARY	INSERT	REPLACE FROM ARRAY	ZAP
COPY MEMO	INSERT AUTOMEM	REPLACE MEMO	
COPY TO ARRAY	ISBLANK()	REPLACE MEMO... FROM	

Table organization

AVERAGE	FOR()	ORDER()	SET UNIQUE
CALCULATE	FOUND()	REINDEX	SET VIEW
CLOSE INDEXES	INDEX	RELATION()	SORT
CONTINUE	JOIN	SEEK	SUM
COPY INDEXES	KEY()	SEEK()	TAG()
COPY TAG	KEYMATCH()	SET FILTER	TAGCOUNT()
CREATE QUERY	LIST	SET IBLOCK	TAGNO()
CREATE VIEW	LOCATE	SET INDEX	TARGET()
CREATE VIEW... FROM ENVIRONMENT	LOOKUP()	SET KEY TO	TOTAL
DELETE TAG	MDX()	SET NEAR	UNIQUE()
DESCENDING()	MODIFY QUERY	SET ORDER	UPDATE
DISPLAY	MODIFY VIEW	SET RELATION	SET UNIQUE
FIND	NDX()	SET SKIP	

Printing

???	_pdriver	_pspacing	PCOL()
_alignment	_peject	_rmargin	PRINTJOB
_box	_pepage	_tabs	PRINTSTATUS()
_indent	_pform	_wrap	PROW()
_lmargin	_plength	CHOOSEPRINTER()	SET MARGIN
_padvance	_plineno	CLOSE PRINTER	SET PCOL
_pageno	_ploffset	DEFINE BOX	SET PRINTER
_pbpage	_porientation	EJECT	SET PROW
_pcolno	_ppitch	EJECT PAGE	
_pcopies	_pquality	ON PAGE	

Input/Output

?	CLEAR GETS	MODIFY LABEL	SET DELIMITERS
??	CLEAR SCREENS	MODIFY REPORT	SET DEVICE
@...CLEAR	CLOSE ALTERNATE	READ	SET FORMAT
@...FILL	CLOSE FORMAT	RELEASE SCREENS	SET HEADINGS
@...SAY...GET	COL()	REPORT FORM	SET SPACE
@...SCROLL	CREATE LABEL	RESTORE SCREENS	TEXT
@...TO	CREATE REPORT	ROW()	UPDATED()
ACCEPT	INPUT	SAVE SCREEN	VARREAD()
ACTIVATE SCREEN	LABEL FORM	SET ALTERNATE	WAIT

dBASE IV Windows

ACTIVATE WINDOW	DEFINE WINDOW	RELEASE WINDOWS	SAVE WINDOW
CLEAR WINDOWS	MOVE WINDOW	RESTORE WINDOW	WINDOW()
DEACTIVATE WINDOW			

dBASE IV Menus

ACTIVATE MENU	DEFINE BAR	ON EXIT POPUP	PADPROMPT()
ACTIVATE POPUP	DEFINE MENU	ON MENU	POPUP()
BAR()	DEFINE PAD	ON PAD	PROMPT()
BARCOUNT()	DEFINE POPUP	ON POPUP	RELEASE MENUS
BARPROMPT()	MENU()	ON SELECTION BAR	RELEASE POPUPS
CLEAR MENUS	ON BAR	ON SELECTION MENU	SHOW MENU
CLEAR POPUPS	ON EXIT BAR	ON SELECTION PAD	SHOW POPUP
DEACTIVATE MENU	ON EXIT MENU	ON SELECTION POPUP	
DEACTIVATE POPUP	ON EXIT PAD	PAD()	

Forms

_curobj	CREATE MENU	MODIFY FORM	ON SELECTION FORM
CLOSE FORMS	CREATE POPUP	MODIFY MENU	OPEN FORM
CREATE APPLICATION	CREATE SCREEN	MODIFY SCREEN	READMODAL()
CREATE FORM	MODIFY APPLICATION	MSGBOX()	SET CUAENTER

Objects

_app	INSPECT()	PLAY SOUND	RESTORE IMAGE
CLASS...ENDCLASS	LISTCOUNT()	REDEFINE	SHOW OBJECT
DEFINE	LISTSELECTED()	RELEASE OBJECT	

Keyboard and mouse

CLEAR TYPEAHEAD	LASTKEY()	ON ESCAPE	SET ESCAPE
FKLABEL()	MCOL()	ON KEY	SET FUNCTION
FKMAX()	MDOWN()	ON MOUSE	SET KEY
INKEY()	MROW()	READKEY()	SET MOUSE
ISMOUSE()	NEXTKEY()	SET CURSOR	SET TYPEAHEAD
KEYBOARD			

Colors and fonts

DEFINE COLOR	GETFONT()	SET COLOR OF	SET COLOR TO
GETCOLOR()	ISCOLOR()		

Environment

CHARSET()	LIST STATUS	SET EDITOR	SET TALK
CLEAR	MEMORY()	SET FULLPATH	SET
CLEAR ALL	SET BELL	SET INTENSITY	SET()
CREATE SESSION	SET BORDER	SET LDCHECK	SETTO()
DISPLAY MEMORY	SET CONFIRM	SET LDCONVERT	SHELL()
DISPLAY STATUS	SET CONSOLE	SET MESSAGE	VERSION()

Environment

LDRIVER()	SET DESIGN	SET ODOMETER
LIST MEMORY	SET DISPLAY	SET SAFETY

Disk and file utilities

!	ERASE	FUNIQUE()	PUTFILE()
_dbwinhome	FACCESSDATE()	GETDIRECTORY()	RENAME
CD	FCREATEDATE()	GETENV()	RUN
COPY FILE	FCREATETIME()	GETFILE()	RUN()
CREATE FILE	FDATE()	HOME()	SET DEFAULT
DELETE FILE	FILE()	LIST FILES	SET DIRECTORY
DIR	FNAMEMAX()	MD	SET PATH
DISKSPACE()	FSHORTNAME()	MKDIR	TYPE
DISPLAY FILES	FSIZE()	MODIFY FILE	VALIDDRIVE()
DOS	FTIME()	OS()	

Low level access

FCLOSE()	FERROR()	FOPEN()	FSEEK()
FCREATE()	FFLUSH()	FPUTS()	FWRITE()
FEOF()	FGETS()	FREAD()	

Preprocessor

#define	#ifdef	#include	#undef
#if	#ifndef	#pragma	

Security

ACCESS()	PROTECT	SET ENCRYPTION	USER()
LOGOUT			

Shared data

BEGINTRANS()	ID()	ON NETERROR	SET LOCK
CHANGE()	LKSYS()	RLOCK()	SET REFRESH
COMMIT()	LOCK()	ROLLBACK()	SET REPROCESS
CONVERT	NETWORK()	SET EXCLUSIVE	UNLOCK
FLOCK()			

Windows programming

BITAND()	BITSET()	HELP	RESOURCE()
BITLSHIFT()	BITXOR()	LOAD DLL	SET HELP
BITOR()	EXTERN	RELEASE DLL	SET TOPIC
BITRSHIFT()			

Classes			
ARRAY	EDITOR	MENUBAR	RADIOBUTTON
ASSOCARRAY	ENTRYFIELD	OBJECT	RECTANGLE
BROWSE	FORM	OLE	SCROLLBAR
CHECKBOX	IMAGE	OLEAUTOCLIENT	SHAPE
COMBOBOX	LINE	PAINTBOX	SPINBOX
DDELINK	LISTBOX	POPUP	TABBOX
DDETOPIC	MENU	PUSHBUTTON	TEXT
Windows 95			
FACCESSDATE()	FCREATETIME()	FNAMEMAX	FSHORTNAME
FCREATEDATE()			



Commands and functions

Commands and functions

! Disk and file utilities

Executes a single DOS command or program from within dBASE.

Syntax

! <DOS command>

<DOS command> A command recognized by your DOS operating system.

Description

! and RUN are equivalent commands. See the description of RUN.

Example

The following examples use ! or RUN to issue the DOS COPY command to copy Clients.DBF and Clients.MDX to a floppy diskette in an external drive:

```
! COPY C:\VISUALDB\SAMPLES\Clients.DBF B:  
RUN COPY C:\VISUALDB\SAMPLES\Clients.MDX B:
```

See Also

DOS, RUN, RUN()

&&

Programs

Marks the characters to the right as a comment rather than as executable code.

Syntax

&& <comment>

<comment> Any sequence of characters on the same line as and following &&. If the last character is a semicolon, dBASE recognizes the following line as a comment.

Description

Use the double ampersand (&&) to make a short comment to the right of a command line. You can use && for a single-line comment; however, programmers generally use the asterisk (*) or NOTE command for single-line comments.

Use comment lines in a program to document the function or logic of a series of commands. If you want to modify the program at a later date, comment lines can identify the sections to be changed.

You can use && after a semicolon that continues a statement to the next line, and you can put a semicolon at the end of a comment to make it continue on the next line. If dBASE finds a semicolon before && on a line, it assumes the next line is a continuation of the command statement. If it finds a semicolon after &&, it assumes the next line is a continuation of the comment.

Example

The following example uses && for adding notes to program files to enhance portability:

```

SET TALK OFF           && Suppresses echoing
CLEAR                 && Clears results pane
SELECT 1              && Selects work area 1
USE Contact ORDER CompCode && Open table with .MDX tag order
SELECT 2              && Work area 2 active
USE Company ORDER CompCode
SET RELATION TO CompCode INTO A && Links work areas A and B (1 and 2)
SET FIELDS TO Company, State_Prov,;
    A->Contact        && Fields in current work area do not;
                        && need alias names.
LIST TO PRINT OFF     && OFF suppresses record numbers.
RETURN
```

See Also

*, NOTE

Marks an entire program line as a comment rather than an executable line.

Syntax

* <comment>

<comment> A sequence of up to 1023 characters on the same line. If the last character is a semicolon, dBASE recognizes the following line as a comment.

Description

Use the asterisk (*) to make an entire program line a comment rather than an executable line. The * must be the first nonblank character on the line. When you execute the program, dBASE ignores the line.

Use comment lines in a program to provide useful notes concerning the function or logic of a series of commands. If you want to alter the program at a later date, comment lines can identify the sections to change. You can also use * in front of a command line to make it a comment temporarily when debugging a program.

To put a comment on the same line as a program statement, use &&.

The * and NOTE are identical in function.

Example

The following example uses an * to add comments in program code:

```
** CLINTRPT.PRG **
USE Clients ORDER Client_ID
* Table ordered by Client_ID .MDX index Tag
SET FIELDS TO Company, Contact
* Creates a view comprised of just two fields
LIST OFF TO PRINT
* Directs output of the selected fields for the entire table to the printer.
* OFF suppresses the printing of Record numbers.
CLOSE DATABASES
```

See Also

&&, NOTE

Evaluates 0 or more expressions and displays or prints the result on a new line.

Syntax

```
?
[<exp 1>
  [PICTURE <format expC>]
  [FUNCTION <function expC>]
  [AT <column expN>]
  [STYLE [<fontstyle expN>] [<fontstyle expC>]]
[,<exp 2>...]
.]
```

<exp 1>[,<exp 2> ...] An expression or expressions of any data type to evaluate.

PICTURE <format expC> Formats <exp 1>, or a specified portion of it, with the picture template <format expC>, a character expression consisting of one of the following:

- Template characters.

- Function symbols preceded by @. (You can also use the FUNCTION option, discussed below.)
- Literal characters.
- A combination of template characters, function symbols, and literal characters.
- A variable containing the character expression.

You can use all the template character and function symbols except A, M, R, and S. For more information about template characters, see Picture in Chapter 8.

FUNCTION *<function expC>* Formats all characters in *<exp 1>* with the function template *<function expC>*, which must contain one or more function symbols. When you specify function symbols with the FUNCTION option, you don't have to precede them with @. For more information about function symbols, see Function in Chapter 8.

AT *<column expN>* Specifies a character column, *<column expN>*, at which the ? command starts displaying or printing *<exp 1>*. The *<column expN>* argument must be between 0 and 255.

STYLE [*<fontstyle expN>*] [*<fontstyle expC>*] Specifies an optional font or style option. Fonts (*<fontstyle expN>*) are specified in the [Fonts] section of DBASEWIN.INI, in a format like the following:

```
[Fonts]
1=Times New Roman,12,ROMAN
2=Courier New,10,MODERN
```

The first argument is the face name of the font; the second argument is the point size; and the third argument is the font family. You can define up to 32,766 font styles. (When assigning numbers greater than 999, don't use whole-number separators.)

If you want to add a font to DBASEWIN.INI but don't know its exact name or family, use GETFONT(). Then add the information GETFONT() returns into DBASEWIN.INI. In addition, the string that GETFONT() returns may be used for *<fontstyle expC>*.

The following table lists the codes you can use for *<fontstyle expC>*:

<i><fontstyle expC></i>	Result
"B"	Bold
"I"	Italic
"S"	Strikeout
"U"	Underline
"R"	Superscript
"L"	Subscript

, The trailing comma is optional and has no effect; it is included for backward compatibility with dBASE IV.

Description

Use ? to display or print (to a file or a printer) the value of valid expressions of any type. Use it like DISPLAY (for example, to display the contents of the current record). In programs, use ? with no expression to skip a line in output.

The ? command differs from ?? only in that ? skips a line before displaying a value, while ?? doesn't.

If SET PRINTER is ON, output from the ? command prints either to the default printer or to the printer or file you've specified with SET PRINTER TO. If SET ALTERNATE is ON, dBASE stores output from the ? command to the file you've specified with SET ALTERNATE TO.

Use the STYLE option with ? or ?? to change the font styles of individual items.

To overstrike a line of text from a program file in printed output, use the AT option with _wrap set to false (.F.). To overwrite rather than overstrike text in printed output, use the AT option with _wrap set to true (.T.), which causes only the second line to print.

To override both an overall _alignment setting and individual paragraph alignments in a memo field, use the B, I, or J functions.

Issue SET SPACE ON (the default) to control whether dBASE inserts a space between expressions in the list when it displays or prints them.

Example

This example uses ? and ?? to display a first and last name in various formats:

```

Firstname="Sally      "
Lastname ="Stephens  "
? Firstname,Lastname
* simple display, no formatting or positioning
* Sally      Stephens

? Firstname PICTURE "@T"
?? " "
?? Lastname PICTURE "@!"
* trim Firstname, make lastname uppercase
* Sally STEPHENS

? Lastname STYLE "B"
?? Firstname AT 20
* display in fixed columns.
* Lastname will print in boldface
* Stephens          Sally

```

This example formats -3273.68 four different ways:

```

N=-3273.68
? N PICTURE "9,999,999.99" && insert commas
*      -3,273.68
? N PICTURE "9,999,999"      && no decimals
*      -3,274
? N PICTURE "@L 9,999,999" && zero fill
* -0003,274
? N PICTURE "@(BT"          && use () for negative number, left-align, and trim

```

??

* (3273.68)

Portability

The AT, PICTURE, FUNCTION, STYLE, and trailing comma options aren't supported in dBASE III PLUS.

See Also

??, _alignment, _wrap, DISPLAY, GETFONT(), LIST, PRINTJOB, SET MEMOWIDTH, SET ALTERNATE, SET PRINTER, SET SPACE, TEXT

??

Input/Output

Evaluates 0 or more expressions and displays or prints the result on the current line.

Syntax

??

```
[<exp 1>
  [PICTURE <format expC>]
  [FUNCTION <function expC>]
  [AT <column expN>]
  [STYLE [<fontstyle expN>] [<fontstyle expC>]]
[,<exp 2>...]
[.]
```

Description

The ?? command is identical to the ? command, except it displays or prints on the current line rather than on a new line. See the ? command for a description of the options of both the ? and ?? commands.

Example

See the example of the ? command, which also contains an example of ??.

See Also

?

???

Printing

Sends output directly to the printer, bypassing the installed printer driver. This command is provided for compatibility with dBASE IV but isn't recommended in *Visual dBASE*.

For complete information on ???, see online Help.

@...CLEAR

Input/Output

Clears a portion of the results pane of the Command window or the current dBASE IV window. This command is supported primarily for compatibility with dBASE IV. In *Visual* dBASE, use DEFINE to create *forms*, which are used instead of dBASE IV windows.

For complete syntax information on @...CLEAR, see online Help. For information about defining forms, see the Forms chapters in the *User's Guide*.

@...FILL

Input/Output

Specifies the colors for a rectangular portion of the results pane of the Command window or the current dBASE IV window. This command is supported primarily for compatibility with dBASE IV. In *Visual* dBASE, use DEFINE to create *forms*, which are used instead of dBASE IV windows.

For complete syntax information on @...FILL, see online Help. For information about defining forms, see the Forms chapters in the *User's Guide*.

@...SAY...GET

Input/Output

Displays or accepts information in a specified format at a specified location in the results pane of the Command window or the current dBASE IV window. This command is supported primarily for compatibility with dBASE IV. In *Visual* dBASE, use DEFINE with the Text and EntryField classes for displaying and accepting information on a form.

For complete syntax information on @...SAY...GET, see online Help. For more information about working with *Visual* dBASE forms, see the Forms chapters in the *User's Guide*.

@...SCROLL

Input/Output

Shifts the contents of a specified region of the results pane of the Command window or the current dBASE IV-style window left, right, up, or down. This command is supported primarily for compatibility with dBASE IV. In *Visual* dBASE, forms have scroll bars the user can use to scroll through a form.

For complete syntax information on @...SCROLL, see online Help. For more information about working with *Visual* dBASE forms, see DEFINE and the Forms chapters in the *User's Guide*.

Draws a box in the results pane of the Command window or the current dBASE IV window. This command is supported primarily for compatibility with dBASE IV. In *Visual* dBASE, use DEFINE to create *forms*, which are used instead of dBASE IV windows.

For complete syntax information on @...TO, see online Help. For information about defining forms, see the Forms chapters in the *User's Guide*.

ABS()

Numeric data

Returns the absolute value of a specified number.

Syntax

ABS(<expN>)

<expN> The numeric or float data type number whose absolute value to return.

Description

ABS() returns the absolute value of a numeric or float data type number. ABS() returns a numeric or float data type according to the data type of the number you supply. The absolute value of a number represents its magnitude. Magnitude is always expressed as a positive value, so the absolute value of a negative number is its positive equivalent.

Example

The following example uses ABS() to take the positive value of a number whatever its sign:

```
? abs(0.2)      && 0.2
? abs( -5)      && 5
? abs(0.2)<1    && true because .2 is less than 1
? abs( -5)<1    && false because 5 is greater than 1
```

See Also

CEILING(), FLOOR(), INT(), ROUND()

ACCEPT

Input/Output

Accepts a user-entered character string and stores it to a memory variable. This command is supported primarily for compatibility with dBASE IV. In *Visual* dBASE, use DEFINE with the Text and EntryField classes for displaying and accepting information on a form.

For complete syntax information on ACCEPT, see online Help. For more information about working with *Visual* dBASE forms, see the Forms chapters in the *User's Guide*.

ACCESS()

Security

Returns the access level of the current user, as assigned with the PROTECT command.

Syntax

ACCESS()

Description

Use ACCESS() to build security into an application. The access level returned can be used to test privileges assigned with PROTECT. If a user is not logged in to the application, ACCESS() returns 0 (zero).

If you write programs that use encrypted files, check the user's access level early in the program. If ACCESS() returns zero, your program might prompt the user to log in, or to contact the system administrator for assistance.

For more information, see PROTECT.

See Also

LOGOUT, PROTECT, SET ENCRYPTION, USER()

ACOPY()

Memory variables

Copies elements from one declared array to another and returns the number of elements copied.

Syntax

ACOPY(<source array name>, <target array name>
[, <starting element expN> [, <elements expN> [, <target element expN>]]])

<source array name> The name of the declared array from which to copy elements.

<target array name> The name of the declared array that elements from <source array name> are copied to.

<starting element expN> The position of the element in <source array name> from which ACOPY() starts copying. Without <starting element expN>, ACOPY() copies all the elements in <source array name> to <target array name>.

<elements expN> The number of elements in <source array name> to copy. Without <elements expN>, ACOPY() copies all the elements in <source array name> from <starting element expN> to the end of the array. If you want to specify a value for <elements expN>, you must also specify a value for <starting element expN>.

<target element expN> The position in <target array name> to which ACOPY() starts copying. Without <target element expN>, ACOPY() copies elements to <target array name> starting at the first position. If you want to specify a value for <target element expN>, you must also specify values for <starting element expN> and <elements expN>.

Description

ACOPY() copies elements from a source array to a target array without regard to the current data types of target positions. If the data type of an element in the source array is different from the data type of the associated element in the target array, ACOPY() overwrites the target value and data type with the source value and data type.

If the target array isn't big enough to contain all the elements being copied from the source array, dBASE displays an error message and does not copy any elements.

Example

The following example uses ACOPY() to create a second array, Ato from the first array, Afrom. The later portion of the example, consisting of a counting loop and the DISPLAY (?) command, sends the results to the Command window results pane for confirmation:

```
PUBLIC Afrom, Ato, Copied
SET TALK OFF
DECLARE Afrom[4],Ato[4]
Afrom[1] = 1
Afrom[2] = "Two"
Afrom[3] = .t.
Afrom[4] = 4
Cnt=1
Copied=ACOPY(Afrom,Ato)
DO WHILE Cnt<=4
    ? Ato[Cnt]
    Cnt=Cnt+1
ENDDO
? "Elements Copied ",Copied
SET TALK ON
```

Portability

Not supported in dBASE IV or dBASE III PLUS.

See Also

AFIELDS(), ALLEN(), APPEND FROM ARRAY, COPY TO ARRAY, DECLARE, REPLACE FROM ARRAY

ACOS()

Numeric data

Returns the inverse cosine (arccosine) of a number.

Syntax

ACOS(<expN>)

<expN> The cosine of an angle, from -1 to +1.

Description

ACOS() returns the radian value of the angle whose cosine is *<expN>*. ACOS() returns a float from 0 to pi radians. ACOS() returns zero when *<expN>* is 1. For values of *x* from 0 to pi, ACOS(*Y*) returns *x* if COS(*X*) returns *y*.

To convert the returned radian value to degrees, use RTOD(). For example, if the default number of decimal places is 2, ACOS(.5) returns 1.05 radians while RTOD(ACOS(.5)) returns 60.00 degrees.

Use SET DECIMALS to set the number of decimal places ACOS() displays.

To find the arcsecant of a value, use 1 divided by the arccosine of the value. For example, the arcsecant of pi is ACOS(1/PI()), or 1.25 radians.

Example

The following example uses ACOS() to find the arc cosine of a set of cosine values:

? ACOS(-1)	&&	3.14
? ACOS(0)	&&	1.57
? ACOS(1)	&&	0.00
? RTOD(ACOS(-1))	&&	180.00
? RTOD(ACOS(0))	&&	90.00
? RTOD(ACOS(1))	&&	0.00

RTOD() converts radians to degrees.

Portability

Not supported in dBASE III PLUS.

See Also

ASIN(), ATAN(), ATN2(), COS(), DTOR(), RTOD(), SET DECIMALS

ACTIVATE MENU

dBASE IV menus

Displays and enables a defined dBASE IV menu bar. This command is supported primarily for compatibility with dBASE IV. In *Visual dBASE*, use DEFINE, OPEN FORM, and READMODAL() to create and activate menus associated with forms.

For complete syntax information on ACTIVATE MENU, see online Help. For information about defining forms, see the Forms chapters in the *User's Guide*.

ACTIVATE POPUP

dBASE IV menus

Displays and enables a dBASE IV pop-up menu. This command is supported primarily for compatibility with dBASE IV. In *Visual dBASE*, use DEFINE, OPEN FORM, and READMODAL() to create and activate menus associated with forms.

For complete syntax information on ACTIVATE POPUP, see online Help. For information about defining forms, see the Forms chapters in the *User's Guide*.

ACTIVATE SCREEN

Input/Output

Sends output to the results pane of the Command window rather than to the current dBASE IV window. This command is supported primarily for compatibility with dBASE IV. In *Visual* dBASE, use DEFINE, OPEN FORM, and READMODAL() to create and activate forms.

For complete syntax information on ACTIVATE WINDOW, see online Help. For information about defining forms, see the Forms chapters in the *User's Guide*.

ACTIVATE WINDOW

dBASE IV windows

Opens dBASE IV-style windows and directs subsequent screen input and output to the last window opened. This command is supported primarily for compatibility with dBASE IV. In *Visual* dBASE, use DEFINE, OPEN FORM, and READMODAL() to create and activate forms.

For complete syntax information on ACTIVATE WINDOW, see online Help. For information about defining forms, see the Forms chapters in the *User's Guide*.

ADEL()

Memory variables

Deletes an element from a one-dimensional array, or deletes a row or column of elements from a two-dimensional array. Returns 1 if successful, an error if unsuccessful.

Syntax

ADEL(<array name>, <position expN> [, <row/column expN>])

<array name> The name of the declared one- or two-dimensional array from which to delete data.

<position expN> When <array name> is a one-dimensional array, <position expN> specifies the number of the element to delete.

When <array name> is a two-dimensional array, <position expN> specifies the number of the row or column whose elements you want to delete. The third argument (discussed in the next paragraph) specifies whether <position expN> is a row or a column.

<row/column expN> Either 1 or 2. If you omit this argument or specify 1, a row is deleted from a two-dimensional array. If you specify 2, a column is deleted. dBASE returns an error if you use <row/column expN> with a one-dimensional array.

Description

Use ADEL() to delete selected elements from an array without changing the size of the array. ADEL() does the following:

- Deletes an element from a one-dimensional array, or deletes a row or column from a two-dimensional array

- Moves all remaining elements toward the beginning of the array (up if a row is deleted, to the left if an element or column is deleted)
- Inserts .F. values in the last position(s)

For information about deleting elements by inserting .F. values and moving remaining elements toward the *end* of the array, see AINS(). For information about replacing elements without moving remaining elements at all, see AFILL(). To change the size of an array, use AGROW() or ARESIZE().

One-dimensional arrays

When you issue ADEL() for a one-dimensional array, the element in the specified position is deleted, and the remaining elements move one position toward the beginning of the array. The logical value .F. is stored to the element in the last position.

For example, if you define a one-dimensional array with DECLARE aArray[3] and STORE "A," "B," and "C" to the array, the array has one row and can be illustrated as follows:

```
A   B   C
```

Issuing ADEL(aArray, 2) deletes element number 2, whose value is "B," moves the value in aArray[3] to aArray[2], and stores .F. to aArray[3] so that the array now contains these values:

```
A   C   .F.
```

Two-dimensional arrays

When you issue ADEL() for a two-dimensional array, the elements in the specified row or column are deleted, and the elements in the remaining rows or columns move one position toward the beginning of the array. The logical value .F. is stored to the elements in the last row or column.

For example, suppose you define a two-dimensional array with DECLARE aArray[3,4] and store letters to the array. The following illustration shows how the array is changed by ADEL(aArray, 2, 2).

Figure 4.1 Using ADEL () with a two-dimensional array

ADEL (aARRAY,2,2)

- 1 Original array created as:
DECLARE aArray[3,4]
STORE "A" TO aArray[1,1]
STORE "B" TO aArray[1,2]
⋮
STORE "L" TO aArray[3,4]

1	2	3	4
A 1,1	B 1,2	C 1,3	D 1,4
5	6	7	8
E 2,1	F 2,2	G 2,3	H 2,4
9	10	11	12
I 3,1	J 3,2	K 3,3	L 3,4

Initial contents of the array aArray

- 2 ADEL(aArray,2,2)
deletes the elements in the
second column...

1	2	3	4
A 1,1	1,2	C 1,3	D 1,4
5	6	7	8
E 2,1	2,2	G 2,3	H 2,4
9	10	11	12
I 3,1	3,2	K 3,3	L 3,4

- 3 Shifts the elements in the
remaining columns towards the
beginning of the array...

1	2	3	4
A 1,1	C 1,2	D 1,3	1,4
5	6	7	8
E 2,1	G 2,2	H 2,3	2,4
9	10	11	12
I 3,1	K 3,2	L 3,3	3,4

- 4 And inserts logical .F. values as
elements in the last column,
resulting in this array:

1	2	3	4
A 1,1	C 1,2	D 1,3	.F. 1,4
5	6	7	8
E 2,1	G 2,2	H 2,3	.F. 2,4
9	10	11	12
I 3,1	K 3,2	L 3,3	.F. 3,4

*Contents of the array after issuing
ADEL(aArray,2,2)*

Example

The following example uses ADEL () and AGROW () to dynamically add and delete to an array which is being edited with @ GET commands:

```
DECLARE aTest[3]  
AFILL(aTest, space(10))  
@0,10 SAY " ALT+A = Add Element ;  
ALT+D = Delete Element"  
ON KEY LABEL ALT+A GrowArray()  
ON KEY LABEL ALT+D DelArray()  
DO WHILE READKEY() <> 12 .and. aTest.size > 0
```

```

    @1,1 CLEAR TO aTest.size, 30
    FOR i = 1 to aTest.size
        @i, 1 SAY i GET aTest[i]
    NEXT
    READ
ENDDO
ON KEY LABEL ALT+A
ON KEY LABEL ALT+D
RETURN

FUNCTION DelArray
    ADEL(aTest, aTest.size)
    KEYBOARD CHR(3)
    RETURN .T.

FUNCTION GrowArray
    AGROW(aTest, 1)
    aTest[aTest.size] = SPACE(10)
    KEYBOARD CHR(3)
    RETURN .T.

```

Portability

Not supported in dBASE IV or dBASE III PLUS.

See Also

AFILL(), AGROW(), AINS(), ARESIZE(), DECLARE

ADIR()

Memory variables

Stores to a declared array five characteristics of specified files: name, size, date stamp, time stamp, and DOS attribute(s). Returns the number of files whose characteristics are stored.

Syntax

```
ADIR(<array name>
[, <filename skeleton expC> [, <DOS file attribute list expC>]])
```

<array name> The name of the declared array of one or more dimensions to which to store the file information. ADIR() dynamically sizes <array name> so the number of rows in the array is equal to the number of files that match <DOS file attribute expC>, and the number of columns is five.

<filename skeleton expC> The file-name pattern (using wildcards) describing the files whose information to store to <array name>.

<DOS file attribute list expC> The letter or letters D, H, S, and/or V representing one or more DOS file attributes.

If you want to specify a value for <DOS file attribute expC>, you must also specify a value or "*" for <filename skeleton expC>.

The meaning of each attribute is as follows:

Character	Meaning
D	Directories
H	Hidden files
S	System files
V	Volume label

If you supply more than one letter for *<DOS file attribute expC>*, include all the letters between one set of quotation marks, for example, ADIR(aArray, "*.PRG", "HS").

Description

Use ADIR() to store information about files to a declared array, which is dynamically resized so all returned information fits in the array.

Without *<filename skeleton expC>*, ADIR() stores information about all files in the current directory that are neither hidden nor system files. For example, if you want to return information only on tables, use "*.DBF" as *<filename skeleton expC>*.

If you want to include directories, hidden files, or system files in the array, use *<DOS file attribute expC>*. When D, H, or S is included in *<DOS file attribute expC>*, all directories, hidden files, and/or system files (respectively) that match *<filename skeleton expC>* are added to the array.

When V is included in *<DOS file attribute expC>*, ADIR() ignores *<filename skeleton expC>* as well as other characters in the attribute list, and stores the volume label to a one-element array.

ADIR() stores the following information for each file into one row of the array. The data type for each is shown in parentheses:

Column 1	Column 2	Column 3	Column 4	Column 5
File name (character)	Size (numeric)	Date (date)	Time (character)	DOS attribute(s) (character)

The last column (DOS attribute) can contain one or more of the following DOS attributes:

Attribute	Meaning
R	Read-only file
A	Archive file (modified since it was last backed up)
S	System file
H	Hidden file
D	Directory

For example, a file with none of the attributes would have the following string in column 5:

.....

A read-only, hidden file would have the following string in column 5:

R..H.

Example

The following example uses ADIR() to store the file name, file size, date of update and time of update for all .DBF files on the current directory to the array Dir_Arr. The counting DO WHILE loop displays the results to the Command window results pane:

```
DECLARE Dir_Arr[1]
Num_Files=ADIR(Dir_Arr,"*.DBF")
Cnt=1
DO WHILE Cnt<=Num_Files
  ? Dir_Arr[Cnt,1], Dir_Arr[Cnt,2] AT 20,;
  Dir_Arr[Cnt,3] AT 35, Dir_Arr[Cnt,4] AT 45,;
  Dir_Arr[Cnt,5] AT 55
  Cnt=Cnt+1
ENDDO
RETURN
```

Portability

Not supported in dBASE IV or dBASE III PLUS.

See Also

ACOPY(), AFIELDS(), ASORT(), CD, DIR, DECLARE, FDATE(), FSIZE(), FTIME()

AELEMENT()

Memory variables

Returns the number of a specified element in a one- or two-dimensional array.

Syntax

AELEMENT(<array name>, <subscript1 expN>
[, <subscript2 expN>])

<array name> A declared one- or two-dimensional array.

<subscript1 expN> The first subscript of the element. In a one-dimensional array, this is the same as the element number. In a two-dimensional array, this is the row.

<subscript2 expN> When <array name> is a two-dimensional array, <subscript2 expN> specifies the second subscript, or column, of the element.

If <array name> is a two-dimensional array and you do not specify a value for <subscript2 expN>, dBASE assumes the value 1. dBASE returns an error if you use <subscript2 expN> with a one-dimensional array.

Description

Use AELEMENT() when you know the subscripts of an element in a two-dimensional array and need the element number for use with another function, such as ACOPY() or ASCAN().

AELEMENT()

In one-dimensional arrays, the number of an element is the same as its subscript, so there is no need to use AELEMENT(). That is, AELEMENT(aOneArray,3) returns 3, AELEMENT(aOneArray,5) returns 5, and so on.

AELEMENT() is the inverse of ASUBSCRIPT(), which returns an element's row or column subscript number when you specify the element number.

Example

The first section of this example initializes a one-dimensional array and a two-dimensional array:

```
DECLARE aTeacher[4]
DECLARE aStudent[3,4]
DISPLAY MEMORY
```

Values held in memory are initialized to logical type and contain the value .F. Note the ordering sequence of the subscripts for the two-dimensional array ASTUDENT:

```
*ATEACHER
*      [ 1]   L .F.
*      [ 2]   L .F.
*      [ 3]   L .F.
*      [ 4]   L .F.
*ASTUDENT
*      [ 1, 1] L .F.
*      [ 1, 2] L .F.
*      [ 1, 3] L .F.
*      [ 1, 4] L .F.
*      [ 2, 1] L .F.
*      [ 2, 2] L .F.
*      [ 2, 3] L .F.
*      [ 2, 4] L .F.
*      [ 3, 1] L .F.
*      [ 3, 2] L .F.
*      [ 3, 3] L .F.
*      [ 3, 4] L .F.
```

The following statements use AELEMENT() to return the number of the element specified by subscripts:

```
? AELEMENT(aTeacher, 3)      && Returns 3
? AELEMENT(aStudent, 1, 2)    && Returns 2
? AELEMENT(aStudent, 2, 2)    && Returns 6
? AELEMENT(aStudent, 3, 4)    && Returns 12
? AELEMENT(aStudent, 3)      && Returns 9
```

Portability

Not supported in dBASE IV or dBASE III PLUS.

See Also

ACOPY(), ADEL(), AFIELDS(), AINS(), ALLEN(), ASCAN(), ASORT(), ASUBSCRIPT(), DECLARE

AFIELDS()

Memory variables

Stores the current table's structural information to a declared array and returns the number of fields whose characteristics are stored.

Syntax

AFIELDS(<array name>)

<array name> The name of a declared array of one or more dimensions.

Description

Use AFIELDS() to store information about the current table structure in a declared array. You can then reference the elements in the array to return information such as a field name and type for use with other functions or for producing reports. Each row in the array contains information on a single field in the current table.

AFIELDS() dynamically sizes <array name> so the number of rows in the array is at least equal to the number of fields in the current table, and the number of columns is at least four. If you declared an array of greater size than required, the rows may not equal the number of fields and the number of columns do not necessarily equal four.

The following table shows which field characteristics AFIELDS() stores, and in which column the information is placed:

Column 1	Column 2	Column 3	Column 4
Field name (character data type)	Field type (character data type)	Field length (numeric data type)	Decimal places (numeric data type)

dBASE uses the following codes for field types: B-dBASE or Paradox binary field (BLOB), C-character, D-date, G-OLE (general), L-logical, M-memo, N-numeric, F-float.

AFIELDS() stores the same information into an array that COPY TO...STRUCTURE EXTENDED stores into a table, except AFIELDS() doesn't create a row containing FIELD_IDX information.

Example

The following example uses AFIELDS() to initialize the array Stru_Arr to the structure of the Company table. The resulting two-dimensional array has four columns containing field name, field type, field length and decimal places and as many rows as the table has fields. The subsequent DO WHILE loop displays the first column only, thus listing the field names of the current table:

```
USE COMPANY
DECLARE Stru_Arr[1]
Num_Fields=AFIELDS(Stru_Arr)
Cnt1=1
DO WHILE Cnt1<=Num_Fields
    ? Stru_Arr[Cnt1,1]
    Cnt1=Cnt1+1
ENDDO
RETURN
```

Portability

Not supported in dBASE IV or dBASE III PLUS.

See Also

COPY TO ARRAY, COPY TO...STRUCTURE EXTENDED, DECLARE, FDATE(), FSIZE(), FTIME()

AFILL()**Memory variables**

Inserts a specified value into one or more locations in a declared array, and returns the number of elements inserted.

Syntax

```
AFILL(<array name>, <exp>
[, <start expN>[, <count expN>]])
```

<array name> The name of a declared one- or two-dimensional array to fill with the specified value *<exp>*.

<exp> An expression of character, date, logical, numeric, or float data type to insert in the specified array.

<start expN> The element number at which to begin inserting *<exp>*. If you do not specify *<start expN>*, dBASE begins at the first element in the array.

<count expN> The number of elements in which to insert *<exp>*, starting at element *<start expN>*. If you do not specify *<count expN>*, dBASE inserts *<exp>* from *<start expN>* to the last element in the array. If you want to specify a value for *<count expN>*, you must also specify a value for *<start expN>*.

If you do not specify *<start expN>* or *<count expN>*, dBASE fills all elements in the array with *<exp>*.

Description

Use AFILL() to insert a value into all or some elements of a declared array. For example, if you are going to use elements of an array to calculate totals, you can use AFILL() to initialize all values in the array to 0.

AFILL() inserts values into the array sequentially. Starting at the first element in the array or at *<start expN>*, AFILL() inserts values in each element in a row, then moves to the first element in the next row, continuing to insert values until the array is filled or until it has inserted *<count expN>* elements. AFILL() overwrites any existing data in the array.

If you know an elements subscripts, you can use AELEMENT() to determine its element number for use as *<start expN>*.

Example

The following example uses AFILL() to replace the current YTD_Sales value held in the 10th column of array Com_Arr. ASCAN() returns the element number for the desired Company name which is used by AFILL() as a reference point:

```
SET TALK OFF
CLEAR
USE Company
Lookup="InterSafe"
Sales=143325552.20
Cnt=RECCOUNT()
Flds=FLDCOUNT()
DECLARE Com_Arr[Cnt,Flds]
COPY TO ARRAY Com_Arr
Element=ASCAN(Com_Arr,Lookup)
IF Element>0
    Rplc=AFILL(Com_Arr,Sales,Element+9,1)
ENDIF
Count=1
DO WHILE Count<=Cnt
    ? Com_Arr[Count,1], Com_Arr[Count,10]
    Count=Count+1
ENDDO
```

Portability

Not supported in dBASE IV or dBASE III PLUS.

See Also

ADEL(), AELEMENT(), AINS(), DECLARE

AGROW()

Memory variables

Adds an element, row, or column to an array and returns a numeric value representing the number of added elements.

Syntax

AGROW (<array name>, <expN>)

<array name> The name of a declared one- or two-dimensional array you want to add elements to.

<expN> Either 1 or 2. When you specify 1, AGROW() adds a single element to a one-dimensional array or a row to a two-dimensional array. When you specify 2, AGROW() adds a column to the array.

Description

Use AGROW() to insert an element, row, or column into an array and change the size of the array to reflect the added elements. AGROW() can make a one-dimensional array two-dimensional. All added elements are initialized to .F. values.

To insert .F. values without changing the size of the array, use AINS().

One-dimensional arrays

When you specify 1 for <expN>, AGROW() adds a single element to the array. When you specify 2, AGROW() makes the array two-dimensional, and existing elements are moved into the first column. This is shown in the following figure:

Figure 4.2 Adding a column to a one-dimensional array using AGROW(bARRAY,2)

AGROW(bARRAY,2)

- 1
- Original array created as:
DECLARE bArray[4]
STORE "A" TO bArray[1]
STORE "B" TO bArray[2]
STORE "C" TO bArray[3]
STORE "D" TO bArray[4]

1	2	3	4
A	B	C	D
1	2	3	4

Initial contents of the array bArray.

- 2
- AGROW(bARRAY,2) adds a new column to the a makes it a two dimensional array with dimension: and copies the old values into the first column.

1	2
A 1,1	.F. 1,2
3	4
B 2,1	.F. 2,2
5	6
C 3,1	.F. 3,2
7	8
D 4,1	.F. 4,2

Contents of the array after issuing AGROW(bArray,2)

Two-dimensional arrays

When you specify 1 for <expN>, AGROW() adds a row to the array and adds the row at the end of the array. This is shown in the following figure:

Figure 4.3 Adding a row to a two-dimensional array using AGROW(aARRAY,1)**AGROW(aARRAY,1)****1** Original array created as:

```

DECLARE aArray[3,4]
STORE "A" TO aArray[1,1]
STORE "B" TO aArray[1,2]
:
STORE "L" TO aArray[3,4]

```

1	A 1,1	2	B 1,2	3	C 1,3	4	D 1,4
5	E 2,1	6	F 2,2	7	G 2,3	8	H 2,4
9	I 3,1	10	J 3,2	11	K 3,3	12	L 3,4

*Initial contents of the array aArray.***2** AGROW(aARRAY,1) adds a new row to the array

1	A 1,1	2	B 1,2	3	C 1,3	4	D 1,4
5	E 2,1	6	F 2,2	7	G 2,3	8	H 2,4
9	I 3,1	10	J 3,2	11	K 3,3	12	L 3,4
13	.F. 4,1	14	.F. 4,2	15	.F. 4,3	16	.F. 4,4

*Contents of the array after issuing
AGROW(aArray,1)*

When you specify 2 for *<expN>*, AGROW() adds a column to the array and places .F. into each element in the column.

Example

The following example initially declares an array of three elements, and then uses AGROW() to add a fourth element, a second column and finally, to add a row to the two dimensional array. DISPLAY MEMORY is used to show the values in the array after each AGROW() operation:

```

RELEASE ALL
DECLARE A[3]
A[1]="x"
A[2]="y"
A[3]="z"
DISPLAY MEMORY
N=AGROW(A,1)    && adds an element to A
DISPLAY MEMORY
N=AGROW(A,2)    && adds a column to A
DISPLAY MEMORY
N=AGROW(A,1)    && adds a new row to A
DISPLAY MEMORY

```

Portability

Not supported in dBASE IV or dBASE III PLUS.

See Also

AINS(), ALLEN(), DECLARE

Inserts an element with the value .F. into a one-dimensional array, or inserts a row or column of elements with the value .F. into a two-dimensional array. Returns 1 if successful, an error if unsuccessful.

Syntax

AINS(<array name>, <position expN> [, <row/column expN>])

<array name> The name of a declared one- or two-dimensional array in which to insert data.

<position expN> When <array name> is a one-dimensional array, <position expN> specifies the number of the element in which to insert an .F. value.

When <array name> is a two-dimensional array, <position expN> specifies the number of a row or column in which to insert .F. values. The third argument (discussed in the next paragraph) specifies whether <position expN> is a row or a column.

<row/column expN> Either 1 or 2. If you omit this argument or specify 1, a row is inserted into a two-dimensional array. If you specify 2, a column is inserted. dBASE returns an error if you use <row/column expN> with a one-dimensional array.

Description

Use AINS() to insert .F. values into selected elements in an array without changing the size of the array. AINS() does the following:

- Inserts an element in a one-dimensional array, or inserts a row or column in a two-dimensional array
- Moves all remaining elements toward the end of the array (down if a row is inserted, to the right if an element or column is inserted)
- Inserts .F. values in the newly created position(s)

For information about inserting elements by moving remaining elements toward the *beginning* of the array and inserting .F. values at the end of the array, see ADEL(). For information about replacing elements without moving remaining elements at all, see AFILL(). To change a one-dimensional array to two-dimensional, use AGROW() or ARESIZE().

One-dimensional arrays

When you issue AINS() for a one-dimensional array, the logical value .F. is inserted into the position of the specified element. The remaining element(s) are moved one place toward the end of the array. The element that had been in the last position is deleted.

For example, if you define a one-dimensional array with DECLARE aArray[3] and STORE "A," "B," and "C" to the array, the array has one row and can be illustrated as follows:

A B C

Issuing `AINS(aArray, 2)` inserts `aArray[2]` with the value `.F.`, moves `aArray[2]`, whose value is `"B,"` to `aArray[3]`, and deletes `"C"` in `aArray[3]` so that the array now contains these values:

A .F. B

Two-dimensional arrays

When you issue `AINS()` for a two-dimensional array, a logical value `.F.` is inserted into the position of each element in the specified row or column. The elements in the remaining columns or rows are moved one place toward the end of the array. The elements that had been in the last row or column are deleted.

For example, suppose you define a two-dimensional array with `DECLARE aArr[3,4]` and store letters to the array. The following figure shows how the array is changed by issuing `AINS(aArray, 2, 2)`:

Figure 4.4 Using AINS() with a two-dimensional array

AINS(aARRAY, 2,2)

- 1
- Original array created as:
DECLARE aArray[3,4]
STORE "A" TO aArray[1,1]
STORE "B" TO aArray[1,2]
:
STORE "L" TO aArray[3,4]

1	A 1,1	2	B 1,2	3	C 1,3	4	D 1,4
5	E 2,1	6	F 2,2	7	G 2,3	8	H 2,4
9	I 3,1	10	J 3,2	11	K 3,3	12	L 3,4

Initial contents of the array aArray

- 2
- AINS(aArray,2,2)
inserts logical .F. values as
elements in the second column...

1	A 1,1	2	B 1,2	3	C 1,3	4	D 1,4
5	E 2,1	6	F 2,2	7	G 2,3	8	H 2,4
9	I 3,1	10	J 3,2	11	K 3,3	12	L 3,4

- 3
- Shifts the elements in the
remaining columns towards the
end of the array, and deletes the
elements from the last column.

1	A 1,1	2	.F. 1,2	3	B 1,3	4	C 1,4
5	E 2,1	6	.F. 2,2	7	F 2,3	8	G 2,4
9	I 3,1	10	.F. 3,2	11	J 3,3	12	K 3,4

- 4
- Resulting in this array:

1	A 1,1	2	.F. 1,2	3	B 1,3	4	C 1,4
5	E 2,1	6	.F. 2,2	7	F 2,3	8	G 2,4
9	I 3,1	10	.F. 3,2	11	J 3,3	12	K 3,4

*Contents of the array after issuing
AINS(aArray,2,2)*

Example

The following example uses a two-dimensional array created as follows:

```
PUBLIC aAlpha
DECLARE aAlpha[2,3]
STORE "one" TO aAlpha[1,1]
STORE "two" TO aAlpha[1,2]
STORE "three" TO aAlpha[1,3]
STORE "four" TO aAlpha[2,1]
STORE "five" TO aAlpha[2,2]
STORE "six" TO aAlpha[2,3]
```

The array aAlpha now contains the following:


```
*aAlpha
* [1,1] C "one"
* [1,2] C "two"
* [1,3] C "three"
* [2,1] C "four"
* [2,2] C "five"
* [2,3] C "six"
```

AINS() is now used to change the first column to .F. and move the remaining elements toward the end of the array:

```
? AINS(aAlpha,1,2)  && Returns 1 if successful
```

aAlpha now contains the following:

```
*aAlpha
* [1,1] C .F.
* [1,2] C "one"
* [1,3] C "two"
* [2,1] C .F.
* [2,2] C "four"
* [2,3] C "five"
```

Portability

Not supported in dBASE IV or dBASE III PLUS.

See Also

ADEL(), AFILL(), AGROW(), ARESIZE(), DECLARE

ALEN()

Memory variables

Returns the number of elements, rows, or columns of an array.

Syntax

ALEN(<array name> [, <expN>])

<array name> The name of a declared one- or two-dimensional array.

<expN> The number 0, 1, or 2, indicating which array information to return: elements, rows, or columns. dBASE returns an error if you specify <expN> for an array that contains more than two dimensions.

The following table describes what ALEN() returns for different <expN> values:

If <expN> is...	ALEN() returns...
not supplied	Number of elements in the array
0	Number of elements in the array
1	For a one-dimensional array, the number of elements For a two-dimensional array, the number of rows (the first subscript of the array)
2	For a one-dimensional array, 0 (zero)

If <expN> is...	ALEN() returns...
	For a two-dimensional array, the number of columns (the second subscript of the array)
any other value	0 (zero)

Description

Use ALEN() to determine the dimensions of a declared array—either the number of elements it contains, or the number of rows or columns it contains.

If you need to determine both the number of rows and the number of columns a two-dimensional array contains, issue ALEN() twice, once with a value of 1 for <expN> and once with a value of 2 for <expN>. For example, issue the following to determine the number of rows and columns contained in aArray:

```
? ALEN(aArray,1) && returns number of rows
? ALEN(aArray,2) && returns number of columns
```

Example

The following example uses ALEN() in a FOR...NEXT loop to print out the contents of an array:

```
USE Clients
DECLARE acContact[RECCOUNT(),1]
COPY TO ARRAY acContact
USE
ShowArray(acContact)
RETURN

FUNCTION ShowArray
PARAMETER avArray
FOR i = 1 to ALEN(avArray)
? avArray[i]
NEXT
RETURN .T.
```

Portability

Not supported in dBASE IV or dBASE III PLUS.

See Also

ADEL(), AELEMENT(), AFIELDS(), AGROW(), AINS(), ARESIZE(), ASCAN(), ASUBSCRIPT(), DECLARE

ALIAS()

Table basics

Returns the alias name of the current or a specified work area. If no table is open in a work area, ALIAS() returns an empty string ("").

Syntax

```
ALIAS([<alias>])
```

<alias> The work area you want to return the alias name of. You can enter a work area number (1 through 225), letter (A through J), or work area name. The work area name can be a table name or an alias name specified with the USE command. The work area letter or alias name must be enclosed in quotes.

Description

ALIAS() returns the alias name of any work area within the current session (if Sessions is checked in the Desktop Properties dialog box). If you do not specify a work area, the current work area is assumed. If no table is opened in the specified work area, ALIAS() returns an empty string ("").

Example

The following example uses ALIAS() to return the name (or alias) of the table currently in use in work area 20:

```
USE Clients ALIAS Customers IN 20
? ALIAS(20)           && Returns CUSTOMERS
? ALIAS("Customers") && Returns CUSTOMERS
SELECT 20
? ALIAS()             && Returns CUSTOMERS
CLOSE DATABASES
```

Portability

Not supported in dBASE III PLUS.

See Also

DBF(), SELECT(), USE, WORKAREA()

ANSI()

String data

Returns a character string that is the American National Standards Institute (ANSI) value of a specified Original Equipment Manufacturer (OEM) character expression.

Syntax

ANSI(<expC>)

<expC> The OEM character expression to convert to ANSI characters.

Description

Use ANSI() and OEM() to convert characters between an ANSI character set and an OEM character set. ANSI() accepts an OEM character expression and converts its characters to ones in the current ANSI character set, while OEM() accepts an ANSI character expression and converts its characters to ones in the current OEM character set. For more information on character sets, see Appendix C in the *Programmer's Guide*.

dBASE uses an OEM character set, while other Windows applications use an ANSI character set. OEM character sets match the ANSI set for most alphabetic and numeric characters but usually differ for high-order characters, those with ASCII values from 128

to 255. You might need to use ANSI() before sending a character string from dBASE to another Windows application, for example, if you are using the EXTERN command.

If you need to export a string with different ANSI and OEM values, use ANSI(). Similarly, if you receive a string from a Windows application, use OEM() to convert it into the format used by dBASE.

The following example shows how you can determine if two characters have the same or different ANSI and OEM values.

Example

The following example displays the 255 characters that are possible with ASCII, ANSI, and OEM formats:

```
FOR i=1 to 255
  ASCII=CHR(i)
  ? i,ASCII,ANSI(ASCII),OEM(ASCII)
  * The next 3 commands cause a pause every 10 lines
  IF MOD(i,10)=0
    WAIT
  ENDIF
NEXT i
```

Portability

Not supported in dBASE IV or dBASE III PLUS.

See Also

CHR(), EXTERN, OEM()

APPEND

Fields and records

Adds new records to a table.

Syntax

```
APPEND
[BLANK]
[NOWAIT]
```

BLANK Adds a blank record to the end of the table and makes the blank record the current record.

NOWAIT Invokes the form to edit a new record but does not move focus there. If used in a program, execution continues to statement following the APPEND NOWAIT command.

Description

APPEND displays the Table Records window in Form layout. Through this window, you can navigate, view, and edit existing records as well as add new records. For more information on editing data and navigating in the Table Records window, see the *User's Guide*.

If you use SET FORMAT to select a format file, APPEND displays record entries as directed by the format file. If no format is specified, *Visual dBASE* displays a default form. The default form displays a single record at a time, with each field displayed left to right.

The APPEND BLANK command adds a blank record to the current table and positions the record pointer on the new record, but it doesn't display a window to edit the data. When accessing SQL tables, some database servers do not allow you to enter blank records. Also, constraints on tables created with non-null fields prevent entering records with fields left blank. For more information, refer to the Borland SQL Link documentation for your particular database server.

Entering data into an appended record and then moving the record pointer past the end of the record causes APPEND to add another record (but not APPEND BLANK). Each new appended record is displayed with blank fields if SET CARRY is OFF. If SET CARRY is ON, each new appended record contains a copy of the data that is in the previous record.

The APPEND command includes only the fields specified in a SET FIELDS TO command, if one is used. If dBASE tables are linked with the SET RELATION command, the CONSTRAIN and INTEGRITY options control operations that add new records to child and parent tables. For more information, see the SET RELATION command.

APPEND automatically updates any open index files while you append records. APPEND displays records in a table with a master index in indexed order and adds the new record to the end of the table.

Example

The following example uses APPEND BLANK to add a blank record at the end of the natural order table:

```
USE Clients
? RECCOUNT()    && Returns number of records in table
APPEND BLANK    && Appends new blank record
? RECCOUNT()    && Returns previous number plus 1
EDIT            && Opens new record for data entry
```

The next example uses APPEND to add a blank record at the end of the current table and open the editing window without having to use the EDIT command, as in the previous example:

```
USE Clients
APPEND
```

See Also

APPEND AUTOMEM, APPEND FROM, BROWSE, CLASS BROWSE, EDIT, SET CARRY, SET FORMAT, SET RELATION, SET WINDOW OF MEMO

APPEND AUTOMEM

Fields and records

Adds a new record to a table using the values stored in automem variables.

Syntax

APPEND AUTOMEM

Description

APPEND AUTOMEM adds a new record to a table and then replaces the value of fields in the table with the contents of corresponding automem variables. Automem variables are variables that have the same names and data types as the fields in the current table.

While the APPEND command presents a display for adding records to a table interactively, APPEND AUTOMEM provides control over data entry in a program. You can customize the editing display and validate data before adding it to a table. APPEND AUTOMEM is also a more efficient way to add new records to a table than using APPEND BLANK and REPLACE.

To use APPEND AUTOMEM to add records to a table, first create a set of automem variables. The USE...AUTOMEM command opens a table and creates the corresponding empty automem variables for that table. CLEAR AUTOMEM creates a set of empty automem variables for the current table or reinitializes existing automem variables to empty values.

When specifying automem variables in commands that allow them (for example, the @...SAY...GET command), you need to prefix the name of an automem variable with m-> to distinguish the variable from fields that may have the same name. For example, @ 0,0 GET Price displays the value in the current record's Price field while @ 0,0 GET m->Price displays the value of the automem variable named Price.

Example

The following example uses APPEND AUTOMEM to add a new record to the Clients table, replacing the value of fields in the table with values held in automem variables. In this case, the new record will be blank because CLEAR AUTOMEM is issued just prior to APPEND AUTOMEM:

```
SET TALK OFF
STORE " " TO Answer
USE Clients
? RECCOUNT()
DO WHILE UPPER(Answer) <> "Q"
    CLEAR AUTOMEM
    APPEND AUTOMEM
    ACCEPT "Press Q to Quit, Enter to Continue " TO Answer
    ? RECCOUNT()
ENDDO
```

Portability

Not supported in dBASE IV or dBASE III PLUS.

See Also

APPEND, CLEAR AUTOMEM, INSERT AUTOMEM, REPLACE AUTOMEM, STORE AUTOMEM, USE

APPEND FROM

Table basics

Copies records from an existing table to the end of the current table.

Syntax

```
APPEND FROM <filename> | ? | <filename skeleton>
[FOR <condition>]
[[TYPE] SDF | PARADOX | DBASE |
  DELIMITED [WITH
    <char> | BLANK]]
[NOVERIFY]
[POSITION]
[REINDEX]
```

<filename> | ? | <filename skeleton> The name of the table whose records you want to append to the current table. APPEND FROM ? and APPEND FROM <filename skeleton> display a dialog box, from which you can select a file. If you specify a file without including its path, *Visual* dBASE first looks for the file in the current directory, then in the path you specify with SET PATH. If you specify a file without including its extension, *Visual* dBASE assumes the default table type specified with the SET DBTYPE command.

You can also append records from a table in a database (defined using the BDE Configuration Utility) by specifying the database as a prefix (enclosed in colons) to the name of the table, that is, *:database name:table name*. If the database is not already open, *Visual* dBASE displays a dialog box in which you specify the parameters, such as a login name and password, necessary to establish a connection to that database.

FOR <condition> Restricts APPEND FROM to records in <filename> that meet <condition>. You can specify a FOR <condition> only for fields that exist in the current table.

[TYPE] SDF | PARADOX | DBASE |

DELIMITED [WITH <char> | BLANK] Specifies the format of data you are appending. The TYPE keyword is included for readability only; it has no effect on the operation of the command. The following table provides a description of the different file formats that are supported:

Type	Description
SDF	A System Data Format file. Records in an SDF file are fixed-length, and the end of a record is marked with a carriage return and a linefeed. In the table receiving SDF data, unused portions of fields are padded with spaces. If you don't specify an extension, <i>Visual</i> dBASE assumes .TXT.
PARADOX	A Paradox table (with a .DB extension). Paradox rows are copied to dBASE records, and each Paradox column is copied to a dBASE field.

Type	Description
DBASE	A dBASE table. If you don't include an extension for <filename>, Visual dBASE assumes a .DBF extension.
DELIMITED	<p>A text format file. Data in the following formats will be translated by Visual dBASE into the current table as follows:</p> <p>Data delimited with quotation marks or the character you specify with WITH <char> is appended to character fields.</p> <p>Data in YYYYMMDD format is appended to date fields.</p> <p>Data consisting solely of the character T or F is appended to logical fields</p> <p>Numbers are appended to numeric fields.</p> <p>Each carriage return and linefeed indicates a new record</p> <p>WITH <char></p> <p>Indicates that <filename> character data is delimited with the character <char> instead of with quotation marks.</p> <p>WITH BLANK</p> <p>Indicates that <filename> data is separated with spaces instead of commas or any other delimiters. If you specify DELIMITED without including WITH BLANK, Visual dBASE assumes that data items in <filename> are separated with commas.</p>

NOVERIFY Disables data validation during APPEND FROM of SDF type files to speed up processing. If you use NOVERIFY, the source data is copied as is into the target fields. Without data validation, Visual dBASE doesn't check or convert source data to fit the data type of target fields.

POSITION Appends fields based on their relative position in the table instead of by matching field names.

REINDEX Rebuilds all open index files after APPEND FROM finishes executing. Without REINDEX, dBASE updates all open indexes after appending each record from <filename>. When the current table has multiple open indexes or contains many records, APPEND FROM executes faster with the REINDEX option.

Description

Use the APPEND FROM command to add data from another file or table to the end of the current table. You can append data from dBASE tables or files in other formats. Data is appended to the current table in the order in which it is stored in the file you specify.

When you specify a dBASE table as the source of data, only the contents of fields whose names and types are the same as the current table are appended. However, Visual dBASE does append date data to a character field with the same name, and appends character data in date format to a date field with the same name.

If the fields common to the current and source tables don't have the same widths, Visual dBASE does one of the following:

- If the field of the current table is longer than the matching source data, Visual dBASE pads the data with spaces.
- If the field of the current table is shorter than the matching field of the source table, Visual dBASE truncates the data.

If SET DELETED is OFF, dBASE adds records from a source dBASE table that are marked for deletion and *doesn't* mark them for deletion in the current table. If SET DELETED is ON, dBASE doesn't add records from a source dBASE table that are marked for deletion.

When importing data from other files, remove column headings and leading blank rows and columns; otherwise, this data is also appended.

Example

The following example uses APPEND FROM to add records to an empty table from an SDF formatted text (.TXT) file:

```
USE Contact ORDER CompCode IN SELECT()
USE Company IN SELECT()
SELECT COMPANY
SET RELATION TO CompCode INTO Contact
COPY STRUCTURE TO CntctLst;
  FIELDS Company->Company, ;
  Contact->CompCode, Contact->Contact, ;
  Company->Street1, Company->Street2, ;
  Company->City, Company->State_Prov,;
  Company->Zip_P_Code
COPY TO CntctLst.TXT TYPE SDF;
  FIELDS Company->Company, ;
  Contact->CompCode, Contact->Contact, ;
  Company->Street1, Company->Street2, ;
  Company->City, Company->State_Prov,;
  Company->Zip_P_Code
CLOSE ALL
USE CntctLst
APPEND FROM CntctLst.TXT TYPE SDF
CLOSE ALL
```

The following example uses APPEND FROM to add records to Clients table from a text file delimited with " (MAILLIST.TXT fields are in the same order as Clients):

```
USE Clients
COPY TO MailList.TXT TYPE DELIMITED
APPEND FROM MailList.TXT TYPE DELIMITED
```

See Also

APPEND, APPEND AUTOMEM, COPY, IMPORT, REINDEX, SET DELETED

APPEND FROM ARRAY

Fields and records

Adds to the current table one or more records containing data stored in a specified array.

Syntax

```
APPEND FROM ARRAY <array name>
[FIELDS <field list>]
[FOR <condition>]
[REINDEX]
```

<array name> The array containing the data to store in the current table as records.

FIELDS <field list> Appends <array name> data only to the fields in <field list>. Without FIELDS <field list>, APPEND FROM ARRAY appends to each field an element from the specified row of <array name>, in the order that the data is stored in the array.

FOR <condition> Restricts APPEND FROM ARRAY to array rows in <array name> that meet <condition>. The FOR <condition> statement can reference a field; *Visual* dBASE recognizes which array element corresponds to which field and evaluates whether to append a given array row as if the array element were the named field.

REINDEX Rebuilds open indexes after all records have been changed.

Description

APPEND FROM ARRAY treats one- and two-dimensional arrays as tables, with columns corresponding to fields and rows corresponding to records. A one-dimensional array works as a table with only one row; therefore, you can append only one record from a one-dimensional array. A two-dimensional array works as a table with multiple rows; therefore, you can append as many records from a two-dimensional array as it has rows. With arrays of more than two dimensions, the last dimension subscript indicates the number of fields to which to add an array row, and the next-to-last dimension subscript indicates the number of records to append. Array subscripts previous to the last two are ignored.

When you append data from an array to the current table, *Visual* dBASE appends each array row as a single record. If the table has more fields than the array has columns, the excess fields are left empty. If the array has more columns than the table has fields, the excess columns are disregarded. The data in the first column is added to the first field's contents, the data in the second column to the second field's contents, and so on.

The data types of the array must match those of corresponding fields in the table you are appending. If the data type of an array element and a corresponding field don't match, *Visual* dBASE returns an error.

If the current table has a memo field, *Visual* dBASE ignores this field. For example, if the second field is a memo field, *Visual* dBASE adds the data in the array's first column to the first field's contents, and the data in the array's second column to the third field's contents.

Use APPEND FROM ARRAY as an alternative to automem variables when you need to transfer data between tables where the structures are similar but the field names are different.

Example

The following example copies selected fields from the Company table to an array, and uses APPEND FROM ARRAY to move a subset of records for State_Prov = California to Temp.DBF:

```

CLOSE ALL
SET SAFETY OFF
USE Company EXCLUSIVE
INDEX ON CompCode TAG CompCode
DECLARE Dup[RECCOUNT(),3]
COPY FIELDS Company, Compcode,State_Prov TO ARRAY Dup
COPY STRUCTURE TO TEMP.DBF WITH PRODUCTION
USE TEMP IN SELECT()
SELECT TEMP
APPEND FROM ARRAY Dup ;
    FIELDS Company, Compcode, State_Prov ;
    FOR State_Prov = "CA" REINDEX
SET FIELDS TO Company, Compcode, State_Prov
GO TOP
BROWSE
RETURN

```

See Also

APPEND AUTOMEM, COPY TO ARRAY, DECLARE, REPLACE FROM ARRAY, STORE AUTOMEM

APPEND MEMO

Fields and records

Copies a text file to a memo field.

Syntax

```

APPEND MEMO <memo field>
FROM <filename> | ? | <filename skeleton>
[OVERWRITE]

```

<memo field> The memo field of the current table to which the contents of *<filename>* are appended.

FROM <filename> | ? | <filename skeleton> Specifies the file to append to the memo field in the current record. APPEND MEMO ? and APPEND MEMO *<filename skeleton>* display a dialog box from which you can select a file. If you specify a file without including its path, *Visual* dBASE looks for the file in the current directory, then in the path you specify with SET PATH. If you specify a file without including its extension, *Visual* dBASE assumes a .TXT extension.

OVERWRITE Erases the contents of the current record memo field before copying the contents of *<filename>*.

Description

Use APPEND MEMO to copy a text file to the current record memo field. You can copy one text file to each memo field of each record in a table. To overwrite the contents of a memo field in the current record, use OVERWRITE.

While dBASE memo fields can contain types of information other than text, binary fields are recommended for storing images, sound, and other user-defined binary information. For more information, see the REPLACE BINARY command.

Example

The following example appends a new, empty record to Clients table, uses APPEND MEMO to place the contents of a text file in the memo field Notes and takes the user to the record in a Table Editor to edit other fields or review the memo field contents:

```
USE Clients
APPEND BLANK
APPEND MEMO Notes FROM C:\VISUALDB\Readme.TXT
BROWSE
```

Portability

Not supported in dBASE III PLUS.

See Also

COPY MEMO, REPLACE BINARY, REPLACE MEMO, REPLACE MEMO...FROM, REPLACE OLE

ARESIZE()

Memory variables

Increases or decreases the size of an array according to the specified dimensions and returns a numeric value representing the number of elements in the modified array.

Syntax

```
ARESIZE(<array name>, <new rows expN>
[, <new cols expN> [, <retain values expN>]])
```

<array name> The name of a declared one- or two-dimensional array whose size you want to increase or decrease.

<new rows expN> The number of rows the resized array should have. *<new rows expN>* must always be a positive, nonzero value.

<new cols expN> The number of columns the resized array should have. *<new cols expN>* must always be 0 or a positive value. If you omit this option, ARESIZE() changes the number of rows in the array and leaves the number of columns the same.

<retain values expN> Determines what happens to the values of the array when rows are added or removed. If you want to specify a value for *<retain values expN>*, you must also specify a value for *<new cols expN>*.

Description

Use ARESIZE() to change the size of a declared array, making it larger or smaller. To determine the number of rows or columns in an existing array, use ALLEN().

If you add or remove columns from the array, you can use *<retain values expN>* to specify how you want existing elements to be placed in the new array. If *<retain values expN>* is zero or isn't specified, ARESIZE() rearranges the elements, filling in the new rows or columns or adjusting for deleted elements, and adding or removing elements at the end of the array, as needed. This is shown in the following two figures.

You are most likely to want to do this if you don't need to refer to existing items in the array; that is, you plan to update the array with new values.

Figure 4.5 Adding a row and a column to a 3x4 array, rearranging elements

ARESIZE(aARRAY,4,5)

- 1 Original array created as:
DECLARE aArray[3,4]
STORE "A" TO aArray[1,1]
STORE "B" TO aArray[1,2]
⋮
STORE "L" TO aArray[3,4]

1	2	3	4
A 1,1	B 1,2	C 1,3	D 1,4
5	6	7	8
E 2,1	F 2,2	G 2,3	H 2,4
9	10	11	12
I 3,1	J 3,2	K 3,3	L 3,4

Initial contents of the array aArray.

- 2 ARESIZE(aARRAY,4,5) adds a new row and column to the array and rearranges the values of the elements.

1	2	3	4	5
A 1,1	B 1,2	C 1,3	D 1,4	E 1,5
6	7	8	9	10
F 2,1	G 2,2	H 2,3	I 2,4	J 2,5
11	12	13	14	15
K 3,1	L 3,2	.F. 3,3	.F. 3,4	.F. 3,5
16	17	18	19	20
.F. 4,1	.F. 4,2	.F. 4,3	.F. 4,4	.F. 4,5

Contents of the array after issuing
ARESIZE(aArray,4,5)

Figure 4.6 Adding a column to a one-dimensional array, rearranging elements

ARESIZE(bARRAY,4,2)

- 1 Original array created as:
DECLARE bArray[4]
STORE "A" TO bArray[1]
STORE "B" TO bArray[2]
STORE "C" TO bArray[3]
STORE "D" TO bArray[4]

1	2	3	4
A 1	B 2	C 3	D 4

Initial contents of the array bArray.

- 2 ARESIZE(bARRAY,4,2) adds a new column to the array, makes it a two dimensional array with dimensions [4,2], and reassigns the values of the elements.

1	2
A 1,1	B 1,2
3	4
C 2,1	D 2,2
5	6
.F. 3,1	.F. 3,2
7	8
.F. 4,1	.F. 4,2

Contents of the array after issuing
ARESIZE(bArray,4,2)

When you use ARESIZE() on a one-dimensional array, you might want the original row to become the first column of the new array. Similarly, when you use ARESIZE() on a two-dimensional array, you might want existing two-dimensional array elements to remain in their original positions. You are most likely to want to do this if you need to

refer to existing items in the array by their subscripts; that is, you plan to add new values to the array while continuing to work with existing values.

If *<retain values expN>* is a nonzero value, ARESIZE() ensures that elements retain their original values. The following two figures repeat the statements shown in the previous two figures, with the addition of a value of 1 for *<retain values expN>*.

Figure 4.7 Adding a row and a column to a 3x4 array, “preserving elements”

ARESIZE(aARRAY,4,5,1)

- 1
- Original array created as:
DECLARE aArray[3,4]
STORE "A" TO aArray[1,1]
STORE "B" TO aArray[1,2]
⋮
STORE "L" TO aArray[3,4]

1	2	3	4
A 1,1	B 1,2	C 1,3	D 1,4
5	6	7	8
E 2,1	F 2,2	G 2,3	H 2,4
9	10	11	12
I 3,1	J 3,2	K 3,3	L 3,4

Initial contents of the array aArray.

- 2
- ARESIZE(aARRAY,4,5,1) adds a new row and column to the array and maintains the values of the elements

1	2	3	4	5
A 1,1	B 1,2	C 1,3	D 1,4	.F. 1,5
6	7	8	9	10
E 2,1	F 2,2	G 2,3	H 2,4	.F. 2,5
11	12	13	14	15
I 3,1	J 3,2	K 3,3	L 3,4	.F. 3,5
16	17	18	19	20
.F. 4,1	.F. 4,2	.F. 4,3	.F. 4,4	.F. 4,5

Contents of the array after issuing ARESIZE(aArray,4,5,1)

Figure 4.8 Adding a column to a one-dimensional array, “preserving elements”

ARESIZE(bARRAY,4,2,1)

- 1
- Original array created as:
DECLARE bArray[4]
STORE "A" TO bArray[1]
STORE "B" TO bArray[2]
STORE "C" TO bArray[3]
STORE "D" TO bArray[4]

1	2	3	4
A 1	B 2	C 3	D 4

Initial contents of the array bArray.

- 2
- ARESIZE(bARRAY,4,2,1) adds a new column to the array and makes it a two-dimensional array with dimensions 4,2. Each existing element is now the first element in its row.

1	2
A 1,1	.F. 1,2
3	4
B 2,1	.F. 2,2
5	6
C 3,1	.F. 3,2
7	8
D 4,1	.F. 4,2

Contents of the array after issuing ARESIZE(bArray,4,2,1)

Example

The following example initially declares an array of three elements, and then uses `ARESIZE()` to resize the array to `A[5]`, `A[5,2]` and finally back to `A[3]`. `DISPLAY MEMORY` is used to show the values in the array after each `ARESIZE()` operation:

```
RELEASE ALL
DECLARE A[3]
A[1]="x"
A[2]="y"
A[3]="z"
DISPLAY MEMORY
N=ARESIZE(A,5)           && A now has 5 elements
A[4]="new1"
A[5]="new2"
DISPLAY MEMORY
N=ARESIZE(A,5,2,1)
* A now has 5 rows and 2 columns.
* The new cols are all set to .t.
* Old values are retained
DISPLAY MEMORY
N=ARESIZE(A,3,1,1)
* A now is back to the original 3 elements
* use:
DISPLAY MEMORY
* N=ARESIZE(A,3,1,0)
* if you don't need the original values
DISPLAY MEMORY
WAIT
```

Portability

Not supported in dBASE IV or dBASE III PLUS.

See Also

`ADEL()`, `AINS()`, `ALEN()`, `DECLARE`

ASC()

Expressions and type conversion

Returns the numeric ASCII value of a specified character.

Syntax

`ASC(<expC>)`

<expC> The character whose ASCII value to return. You can specify more than one character, but dBASE uses only the first one.

Description

`ASC()` is the inverse function of `CHR()`. `ASC()` accepts a character and returns its ASCII value—a number from 0 to 255, inclusive. `CHR()` accepts an ASCII value and returns its character.

You can use ASC() to manipulate the returned ASCII values of characters with mathematical operators. See the ASCII table in Appendix E for a listing of ASCII values and their corresponding characters.

Example

The following example uses ASC() to determine the ASCII value of a specified character:

```
SET TALK OFF
plus_minus = CHR(241)
? "The ASCII value of " + plus_minus + " is "
?? + LTRIM(STR(ASC(plus_minus),3,0))
```

See Also

ANSI(), CHR(), OEM()

ASCAN()

Memory variables

Searches an array for an expression. Returns the number of the first element that matches the expression if the search is successful, or 0 if the search is unsuccessful.

Syntax

```
ASCAN(<array name>, <exp>
[, <starting element expN> [, <elements expN>]])
```

<array name> A declared one- or two-dimensional array.

<exp> The expression to search for in *<array name>*.

<starting element expN> The element number in *<array name>* at which to start searching. Without *<starting element expN>*, ASCAN() starts searching at the first element.

<elements expN> The number of elements in *<array name>* that ASCAN() searches. Without *<elements expN>*, ASCAN() searches *<array name>* from *<starting element expN>* to the end of the array. If you want to specify a value for *<elements expN>*, you must also specify a value for *<starting element expN>*.

Description

Use ASCAN() to search an array for the value contained in *<exp>*. For example, if an array contains customer names, you can use ASCAN() to find the location in which a particular name appears.

ASCAN() returns the element number of the first element in the array that matches *<exp>*. If you want to determine the subscripts of this element, use ASUBSCRIPT().

When *<exp>* contains string data, ASCAN() is case-sensitive; you may want to use UPPER(), LOWER(), or PROPER() to match the case of *<exp>* with the case of the data stored in the array.

When *<exp>* contains string data, ASCAN() searches for an expression following the rules established by SET EXACT. If SET EXACT is ON, dBASE returns 0 if the value in

<exp> is not identical to the data in an element of the array. If SET EXACT is OFF, dBASE returns 0 if the characters in *<expN>* do not match the beginning characters in the data in an element of the array. The following code example illustrates this more clearly. For more information, see SET EXACT.

```

DECLARE aArray[3,4]                && 3 rows,4 columns
? AFILL(aArray,"abcd",6,1)         && place "abcd" in the 6th element
SET EXACT OFF
? ASCAN(aArray,"abcd")             && returns 6
? ASCAN(aArray,"abc")              && returns 6
? ASCAN(aArray,"bcd")              && returns 0
SET EXACT ON
? ASCAN(aArray,"abcd")             && returns 6
? ASCAN(aArray,"abc")              && returns 0
? ASCAN(aArray,"abc")              && returns 0

```

Example

The following example uses ASCAN() to return an element number for a desired string within an array and ASUBSCRIPT() to return the row and column coordinates within the array:

```

CLEAR
DECLARE A_Dir[1]
FileName="CLIENTS.DBF"
Files=ADIR(A_Dir,"*.")             && Initializes array to directory contents
Asort=ASORT(A_Dir)                 && Orders array
Element=ASCAN(A_Dir,FileName)       && Returns filename location
IF Element > 0                     && ASCAN() returns 1 if successful
    Row=ASUBSCRIPT(A_Dir,Element,1)
    Col=ASUBSCRIPT(A_Dir,Element,2)
    ? "Name" AT 15, "Bytes" AT 30, "Date" AT 39,;
      "Time" AT 48
    ?
    ? "File Info: " + A_Dir[Row,Col];
    + " " +STR(A_Dir[Row,Col+1]);
    + " " +DTOC(A_Dir[Row,Col+2])+" " +A_Dir[Row,Col+3]
ENDIF

```

Portability

Not supported in dBASE IV or dBASE III PLUS.

See Also

ACOPY(), AFIELDS(), AFILL(), ASORT(), ASUBSCRIPT(), DECLARE, LOWER(), PROPER(), SET EXACT, UPPER()

ASIN()

Numeric data

Returns the inverse sine (arcsine) of a number.

Syntax

ASIN(<expN>)

<expN> The sine of an angle, from -1 to +1.

Description

ASIN() returns the radian value of the angle whose sine is <expN>. ASIN() returns a float from $-\pi/2$ to $\pi/2$ radians. ASIN() returns zero when <expN> is 0. For values of x from $-\pi/2$ to $\pi/2$, ASIN(Y) returns x if SIN(X) returns y .

To convert the returned radian value to degrees, use RTOD(). For example, if the default number of decimal places is 2, ASIN(.5) returns .52 radians while RTOD(ASIN(.5)) returns 30.00 degrees.

Use SET DECIMALS to set the number of decimal places ASIN() displays.

To find the arccosecant of a value, use 1 divided by the arcsine of the value. For example, the arccosecant of 1.54 is ASIN(1/1.54), or .71 radians.

Example

The following example uses ASIN() to return the arcsine of a set of sine values:

```

CLEAR
? "ArcSine" AT 20
?
? "Sine" AT 9, "Radians" AT 20, "Degrees" AT 33
SET DECIMALS TO 2
FOR sine = -1 TO 1 STEP .25
    ? sine AT 0, ASIN(sine) AT 13, RTOD(ASIN(sine)) AT 26
NEXT

```

Portability

Not supported in dBASE III PLUS.

See Also

ACOS(), ATAN(), ATN2(), DTOR(), RTOD(), SET DECIMALS, SIN()

ASORT()

Memory variables

Sorts the elements in a one-dimensional array or the rows in a two-dimensional array, returning 1 if successful or an error if unsuccessful.

Syntax

ASORT(<array name>

[, <starting element expN> [, <elements to sort expN> [, <sort order expN>]]])

<array name> A declared one- or two-dimensional array.

<starting element expN> In a one-dimensional array, the number of the element in *<array name>* at which to start sorting. In a two-dimensional array, the number (subscript) of the column on which to sort. Without *<starting element expN>*, ASORT() starts sorting at the first element or column in the array.

<elements to sort expN> In a one-dimensional array, the number of elements to sort. In a two-dimensional array, the number of rows to sort. Without *<elements to sort expN>*, ASORT() sorts the rows starting at the row containing element *<starting element expN>* to the last row. If you want to specify a value for *<elements to sort expN>*, you must also specify a value for *<starting element expN>*.

<sort order expN> The sort order:

- 0 specifies ascending order (the default)
- 1 specifies descending order

If you want to specify a value for *<sort order expN>*, you must also specify values for *<elements to sort expN>* and *<starting element expN>*.

Description

ASORT() succeeds in sorting when all elements you specify to be sorted are of the same data type. The elements to sort in a one-dimensional array must be of the same data type, and the elements of the column by which rows are to be sorted in a two-dimensional array must be of the same data type.

ASORT() arranges elements in alphabetical, numerical, chronological, or logical order, depending on the data type of *<starting element expN>*. (For character data, the current language driver determines the sort order.)

One-dimensional arrays

Suppose you issue DECLARE aNums[8] and store numbers to the array so that the array elements are in this order:

5 7 3 9 4 1 2 8

If you issue ASORT(aNums, 1, 5), dBASE sorts the first five elements so that the array elements are in this order:

3 4 5 7 9 1 2 8

If you then issue ASORT(aNums, 5, 2), dBASE sorts two elements starting at the fifth element so that the array elements are now in this order:

3 4 5 7 1 9 2 8

Two-dimensional arrays

Using ASORT() with a two-dimensional array is similar to using the SORT command with a table. In this comparison, array rows correspond to records, and array columns correspond to fields.

When you sort a two-dimensional array, whole rows are sorted, not just the elements in the column where *<starting element expN>* is located.

ASORT()

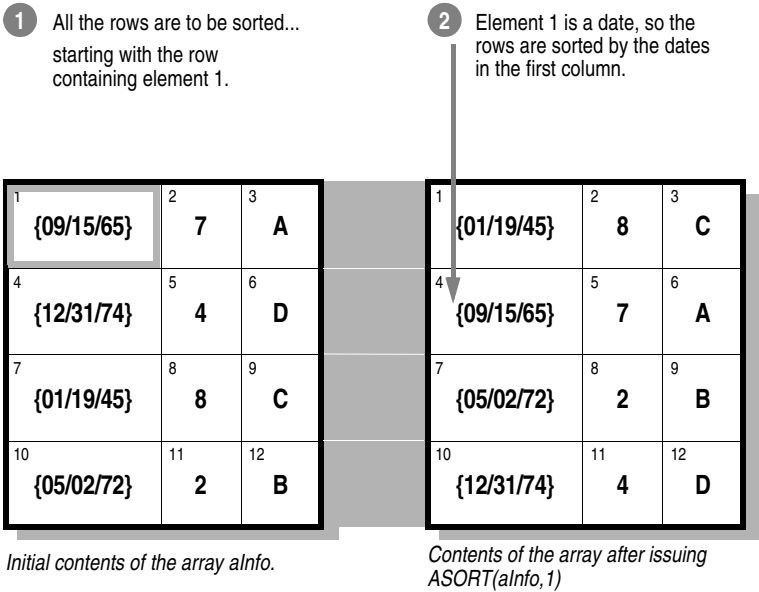
For example, suppose you issue DECLARE aInfo[4, 3] and fill the array with the following data:

{09/15/65}	7	A
{12/31/65}	4	D
{01/19/45}	8	C
{05/02/72}	2	B

If you issue ASORT(aInfo, 1), dBASE sorts all rows in the array beginning with element number 1. The rows are sorted by the dates in the first column because element 1 is a date. The following figure shows the results.

Figure 4.9 ASORT (aInfo,1)

ASORT (aInfo,1)



If you then issue ASORT(aInfo, 5, 2), dBASE sorts two rows in the array starting with element number 5, whose value is 7. ASORT() sorts the second and the third rows based on the numbers in the second column. The following figure shows the results.

Figure 4.10 Using ASORT() with a two-dimensional array

ASORT(alnfo,5,2)

- 1 Two rows are to be sorted (ASORT(alnfo,5,2)) starting with the row containing element 5 (ASORT(alnfo,5,2)).
- 2 Element 5 contains a number, so the rows are sorted by the numbers in the second column.

1	{01/19/45}	2	8	3	C
4	{09/15/65}	5	7	6	A
7	{05/02/72}	8	2	9	B
10	{12/31/74}	11	4	12	D

Initial contents of the array alnfo.

1	{01/19/45}	2	8	3	C
4	{05/02/72}	5	2	6	B
7	{09/15/65}	8	7	9	A
10	{12/31/74}	11	4	12	D

Contents of the array after issuing ASORT(alnfo,5,2)

Example

See ASCAN() for an example of ASORT().

Portability

Not supported in dBASE IV or dBASE III PLUS.

See Also

AELEMENT(), ALEN(), ASCAN(), ASUBSCRIPT(), DECLARE

ASUBSCRIPT()

Memory variables

Returns the row number or the column number of a specified element in an array.

Syntax

ASUBSCRIPT(<array name>, <element expN>, <row/column expN>)

<array name> A declared one- or two-dimensional array.

<element expN> The element number.

<row/column expN> A number, either 1 or 2, that determines whether you want to return the row or column subscript of an array. If <row/column expN> is 1, ASUBSCRIPT() returns the number of the row subscript. If <row/column expN> is 2, ASUBSCRIPT() returns the number of the column subscript.

ASUBSCRIPT()

If *<array name>* is a one-dimensional array, dBASE returns an error if *<row/column expN>* is a value other than 1.

Description

Use ASUBSCRIPT() when you know the number of an element in a two-dimensional array and want to reference the element by using its subscripts.

If you need to determine both the row and column number of an element in a two-dimensional array, issue ASUBSCRIPT() twice, once with a value of 1 for *<row/column expN>* and once with a value of 2 for *<row/column expN>*. For example, if the element number is 13, issue the following to return its subscripts:

```
? ASUBSCRIPT(aArray,13,1) && returns row subscript
? ASUBSCRIPT(aArray,13,2) && returns column subscript
```

In one-dimensional arrays, the number of an element is the same as its subscript, so there is no need to use ASUBSCRIPT(). That is, ASUBSCRIPT(aOneArray,3,1) returns 3, ASUBSCRIPT(aOneArray,5,1) returns 5, and so on.

ASUBSCRIPT() is the inverse of AELEMENT(), which returns an element number when you specify the element subscripts.

Example

The following example uses ASCAN() to return an element number for a desired string within an array and ASUBSCRIPT() to return the row and column coordinates within the array:

```
CLEAR
DECLARE A_Dir[1]
FileName="CLIENTS.DBF"
Files=ADIR(A_Dir,"*.**")      && Initializes array to;
                               directory contents
Asort=ASORT(A_Dir)            && Orders array
Element=ASCAN(A_Dir,FileName) && Returns filename;
                               location
IF Element > 0                && ASCAN() returns 1;
                               if successful
    Row=ASUBSCRIPT(A_Dir,Element,1)
    Col=ASUBSCRIPT(A_Dir,Element,2)
    ? "Name" AT 15, "Bytes" AT 30, "Date" AT 39,;
      "Time" AT 48
    ?
    ? "File Info: " + A_Dir[Row,Col];
      + " "+STR(A_Dir[Row,Col+1]);
      + " "+A_Dir[Row,Col+2]+" "+A_Dir[Row,Col+3]
ENDIF
```

Portability

Not supported in dBASE IV or dBASE III PLUS.

See Also

ACOPY(), ADEL(), AELEMENT(), AFIELDS(), AINS(), ALen(), ASCAN(), ASORT(), DECLARE

AT()

String data

Returns a number that represents the position of a string within another string or memo field.

Syntax

AT(<search expC>, <target expC> | <target memo field>
[, <nth occurrence expN>])

<search expC> The string to search for in <target expC> or <target memo field>.

<target expC> | <target memo field> The string or memo field in which to search for <search expC>.

<nth occurrence expN> Which occurrence of the string to find. By default, dBASE searches for the first occurrence. You can search for other occurrences by specifying the number, which must be greater than zero.

Description

AT() returns the numeric position where a *search string* begins in a *target string* or target memo field. AT() searches, one character at a time, from the first character of the string or memo field, searching left to right, beginning to end.

The search is case-sensitive. Use UPPER() or LOWER() to make the search case-insensitive.

You can specify which occurrence of the search string to find by specifying a number for <nth occurrence expN>. If you omit this argument, AT() returns the starting position of the first occurrence of the search string.

AT() returns 0 when

- The search string isn't found.
- The search string, target string, or target memo field is empty.
- The search string is longer than the target string.
- The <nth occurrence expN> occurrence doesn't exist.

If <nth occurrence expN> is less than 1, dBASE returns an error.

When AT() counts characters in a memo field, it counts two characters for each carriage-return and linefeed combination (CR/LF) in the memo field.

Use RAT() to find the starting position of <search expC>, searching from *right to left*, end to beginning. Use the substring operator (\$) to learn if one string exists within another. See Chapter 1 for information about operators.

Example

The following example uses AT() to determine the starting point of a passed text string in a second string:

```
? AT("B", "ABC")           && Returns 2
? AT("ss", "Mississippi")  && Returns 3
? AT("ss", "Mississippi", 2) && Returns 6
```

ATAN()

```
? AT("Z", "ABC")           && Returns 0
? AT("a", "ABC")           && Returns 0
? AT("ABC", "AB")          && Returns 0
? AT("", "ABC")            && Returns 0
? AT("a", "abc", 2)        && Returns 0
```

The following example uses AT() to display or print a listing of all records in the Client table that contain a string in the Notes memo field and the position of the first occurrence.

```
USE Clients
String="Special Handling"
LIST Company, Notes FOR AT(String, Notes)>0
CLOSE ALL
```

Portability

dBASE IV limits a memo field search to the first 64K of data, and the *<memo field>* and *<nth occurrence expN>* arguments aren't supported in dBASE III PLUS.

See Also

RAT(), STUFF(), SUBSTR()

ATAN()

Numeric data

Returns the inverse tangent (arctangent) of a number.

Syntax

ATAN(*<expN>*)

<expN> Any positive or negative number representing the tangent of an angle.

Description

ATAN() returns the radian value of the angle whose tangent is *<expN>*. ATAN() returns a float from $-\pi/2$ to $\pi/2$ radians. ATAN() returns 0 when *<expN>* is 0. For values of x from $-\pi/2$ to $\pi/2$, ATAN(Y) returns x if TAN(X) returns y .

To convert the returned radian value to degrees, use RTOD(). For example, if the default number of decimal places is 2, ATAN(1) returns 0.79 radians, while RTOD(ATAN(1)) returns 45.00 degrees.

Use SET DECIMALS to set the number of decimal places ATAN() displays.

ATAN() differs from ATN2() in that ATAN() takes the tangent as the argument while ATN2() takes the sine and cosine as the arguments.

To find the arccotangent of a value, subtract $\pi/2$ from the arctangent of the value. For example, the arccotangent of $\pi/3$ is $\pi/2 - \text{ATAN}(\pi/3)$, or .76.

Example

The following example uses ATAN() to find the arctangent of a set of tangent values:


```
CLEAR
SET DECIMALS TO 5
? "Arc Tangent" AT 18
? "Tangent" AT 6, "Radians" AT 20, "Degrees" AT 33
FOR tang = -100 TO 1000 STEP 100
    ? tang AT 0, ATAN(tang) AT 12, ;
    LTRIM(STR(RTOD(ATAN(tang)),8,5)) AT 33
NEXT
```

Portability

Not supported in dBASE III PLUS.

See Also

ACOS(), ASIN(), ATN2(), RTOD(), SET DECIMALS, TAN()

ATN2()

Numeric data

Returns the inverse tangent (arctangent) of a given point.

Syntax

ATN2(<sine expN>, <cosine expN>)

<sine expN> The sine of an angle. If <sine expN> is 0, <cosine expN> can't also be 0.

<cosine expN> The cosine of an angle. If <cosine expN> is 0, <sine expN> can't also be 0. When <cosine expN> is 0 and <sine expN> is a positive or negative (nonzero) number, ATN2() returns +pi/2 or -pi/2, respectively.

Description

ATN2() returns the angle size in radians when you specify the sine and cosine of the angle. ATN2() returns a float from -pi to +pi radians. ATN2() returns 0 when <sine expN> is 0. When you specify 0 for both arguments, dBASE returns an error.

To convert the returned radian value to degrees, use RTOD(). For example, if the default number of decimal places is 2, ATN2(1,0) returns 1.57 radians while RTOD(ATN2(1,0)) returns 90.00 degrees.

Use SET DECIMALS to set the number of decimal places ATN2() displays.

ATN2() differs from ATAN() in that ATN2() takes the sine and cosine as the arguments while ATAN() takes the tangent as the argument. See ATAN() for instructions on finding the arccotangent.

Example

The following example shows some ways to use ATN2():

```
? ATN2(2.79,-3.2)      && Returns 2.42
? ATN2(-3,3)          && Returns -0.79
? RTOD(ATN2(-3,3))    && Returns -45
? RTOD(ATN2(-1,-SQRT(3))) && Returns -150
x=SIN(DTOR(30))
```

```
y=COS(DTOR(30))
? RTOD(ATN2(x,y))      && Returns 30
```

Portability

Not supported in dBASE III PLUS.

See Also

ACOS(), ASIN(), ATAN(), COS(), RTOD(), SET DECIMALS, SIN(), TAN()

AVERAGE

Table organization

Computes the arithmetic mean (average) of specified numeric or float type fields in the current table.

Syntax

```
AVERAGE
[<exp list>]
[<scope>]
[FOR <condition 1>]
[WHILE <condition 2>]
[TO <memvar list> | TO ARRAY <array name>]
```

<exp list> The numeric or float fields, or expressions involving numeric or float fields, to average.

<scope> The range of records to average. RECORD <n> identifies a single record by its record number. NEXT <n> identifies *n* records, beginning with the current record. ALL specifies all records. REST specifies all records from the current record to the end of the file.

FOR <condition 1>

WHILE <condition 2> Determines which records are affected by AVERAGE. FOR restricts AVERAGE to records that meet <condition 1>, starting at the first record of the table or view. WHILE starts processing with the current record and continues with each subsequent record as long as <condition 2> is true.

TO <memvar list> | TO ARRAY <array name> Initializes and stores averages to the memory variables of <memvar list> or stores averages to the existing array <array name>. If you specify an array, each field average is stored to elements in the order in which you specify the fields in <exp list>. If you don't specify <exp list>, each field average is stored in field number order. <array name> can be a single- or multidimensional array. You can store averages to an array without the ARRAY keyword if you explicitly reference the array subscripts.

Description

The AVERAGE command computes the arithmetic means (averages) of numeric expressions and stores the results in specified memory variables or array elements. If you store the values in memory variables, the number of memory variables must be exactly the same as the number of fields or expressions averaged. If you store the values

in an array, the array must already exist, and the array must contain at least as many elements as the number of averaged expressions.

If SET TALK is ON, AVERAGE also displays its results in the results pane of the Command window. The SET DECIMALS setting determines the number of decimal places that AVERAGE displays. Numeric fields in blank records are evaluated as zero. To exclude blank records, use the ISBLANK() function in defining a FOR condition. EMPTY() excludes records in which a specified expression is either 0 or blank.

Example

The following example uses AVERAGE to calculate the average year to date sales for all companies in the Company table:

```
USE Company
AVERAGE Ytd_sales TO Ytd_Av
? "The average Ytd Sale was $", Ytd_av PICTURE "99,999,999.99"
```

Portability

Not supported in dBASE III PLUS.

See Also

CALCULATE, COUNT, SUM, TOTAL

BAR()

dBASE IV menus

Returns the number of the currently selected (highlighted) or most recently chosen bar in a dBASE IV pop-up menu. This command is supported primarily for compatibility with dBASE IV. In *Visual dBASE*, use INSPECT() to return information associated with objects in forms.

For complete syntax information on BAR(), see online Help. For information about defining forms, see the Forms chapters in the *User's Guide*.

BARCOUNT()

dBASE IV menus

Returns the number of bars in a dBASE IV pop-up menu. This command is supported primarily for compatibility with dBASE IV. In *Visual dBASE*, use INSPECT() to return information associated with objects in forms.

For complete syntax information on BARCOUNT(), see online Help. For information about defining forms, see the Forms chapters in the *User's Guide*.

BARPROMPT()

Returns the prompt of a bar in a dBASE IV pop-up menu. This command is supported primarily for compatibility with dBASE IV. In *Visual dBASE*, use INSPECT() to return information associated with objects in forms.

For complete syntax information on BARPROMPT(), see online Help. For information about defining forms, see the Forms chapters in the *User's Guide*.

BEGINTRANS()

Starts a transaction and returns .T. if the transaction started successfully.

Syntax

BEGINTRANS([<database name expC>] [<isolation level expN>])

- <database name expC>** The name of the SQL database in which to begin the transaction.
- If <database name expC> is omitted but a SET DATABASE statement has been issued, BEGINTRANS() refers to the database in the SET DATABASE statement.
 - If <database name expC> is omitted and no SET DATABASE statement has been issued, BEGINTRANS() refers to the database opened after issuing BEGINTRANS().
- <isolation level expN>** Specifies a pre-defined server-level transaction isolation scheme.
- Valid values for <isolation level> are:

expN	Description
0	Server's default isolation level
1	Uncommitted changes read (dirty read)
2	Committed changes read (read committed)
3	Full read repeatability (repeatable read)

- <isolation level> is not supported for local tables.
- If an invalid value is given for <isolation level>, a dBASE "Value out of range" error is generated.
- The <isolation level> is server-specific; a "Not supported" error will result from the database engine if an unsupported level is specified.

Note If you include <database name expC> when you issue BEGINTRANS(), you must also include it in subsequent COMMIT() or ROLLBACK() statements within that transaction. If you don't, dBASE ignores the COMMIT() or ROLLBACK() statement.

Description

Use BEGINTRANS() to initiate a *transaction* during which the user might make changes to a table or tables in an SQL database that supports transaction processing. Within a

transaction initiated with BEGINTRANS(), you can work in only one database. Also, you can't nest transactions.

If you issue BEGINTRANS() against an SQL database that does not support transactions, or if a server error occurs, BEGINTRANS() returns .F. Otherwise, it returns .T. If BEGINTRANS() returns .F., use SQLERROR() or SQLMESSAGE() to determine the nature of the server error that might have occurred.

To close a transaction, use COMMIT() or ROLLBACK(). COMMIT() saves changes made during the transaction, and ROLLBACK() discards changes and returns tables to the state they were in before BEGINTRANS() was issued.

BEGINTRANS() applies to the following commands:

@...SAY...GET	DELETE	REPLACE
APPEND	EDIT	REPLACE MEMO/BINARY/OLE
APPEND BLANK	FLOCK()	RLOCK()
APPEND MEMO	INSERT	
BROWSE	RECALL	

The following commands are not allowed in transactions. dBASE returns an error if you try to issue them from within a transaction.

BEGINTRANS() (nested transactions are not allowed)	DELETE TAG
CLEAR ALL	INDEX
CLOSE ALL/DATABASE/INDEX (any command that closes open tables or indexes)	MODIFY STRUCTURE
CONVERT	PACK
CREATE FROM	a USE that would close an open table, or open a table in another database
ZAP	

Example

The following example begins a transaction with BEGINTRANS(). It opens a multi-user version of Company.dbf and attempts to make all YTD_SALES 0. ON ERROR detects any error which might occur. In particular, it will detect if another user has locked any record in Company.dbf. If an error occurs, ROLLBACK() resets all values. Otherwise COMMIT() writes the changes to disk:

```

CLOSE ALL
SET PROCEDURE TO PROGRAM(1) ADDITIVE
SET EXCLUSIVE OFF

BEGINTRANS()

TransErr=.f.
```

```
BINTYPE( )

ON ERROR DO TransErr    && Activates ON ERROR trap

USE L:\MultiUse\Company
REPLACE ALL Ytd_Sales WITH 0
ON ERROR                && Disables ON ERROR

IF TransErr
    ? "Rollback"
    ROLLBACK()           && restore data
ELSE
    ? "Commit"
    COMMIT()             && save changes
ENDIF

PROC TransErr
WAIT "Warning: Transaction Fails"
TransErr=.t.
```

Portability
Not supported in dBASE IV or dBASE III PLUS. BEGINTRANS() replaces the BEGIN TRANSACTION and END TRANSACTION commands in dBASE IV.

See Also
COMMIT(), FLOCK(), RLOCK(), ROLLBACK(), SET EXCLUSIVE, SQLERROR(), SQLMESSAGE()

BINTYPE()

Fields and records

Returns the predefined type number of a specified binary field.

Syntax
BINTYPE([<field name>])
<field name> The name of a field in the current table.

Description
BINTYPE() returns the predefined binary type number of a binary field in the current table. Using this command, you can determine the type of data stored in the field. The values returned by BINTYPE() are the following:

Predefined binary type numbers	Description
1 to 32K – 1 (1 to 32,767)	User-defined file types
32K (32,768)	.WAV files
32K + 1 (32,769)	.BMP and .PCX files

BINTYPE() returns an error if a non-binary field is specified. It returns a value of 0 if the binary field is empty.

Example

The following example uses BINTYPE() to return the binary type code for the BMP field of the Animals table:

```
USE Animals
GO 5
? BINTYPE(BMP)           && Returns 32769
```

See Also

COPY BINARY, PLAY SOUND REPLACE BINARY, RESTORE IMAGE

BITAND()

Windows programming

Performs a bitwise AND operation on two numeric values and returns the result.

Syntax

```
BITAND(<expN1>, <expN2>)
```

Description

BITAND() is used by advanced programmers to make binary interpretations of numeric values. These values are often returned by functions prototyped with EXTERN to access resources in Windows API or other DLL files. Interpreting such values often requires analysis and manipulation of individual bits.

BITAND() compares bits in the numeric value <expN1> with corresponding bits in the numeric value <expN2>. When two bits in the same position are on (set to 1), the corresponding bit in the returned value is on. In any other case, the bit is off (set to 0).

	Bit 1 = 1	Bit 1 = 0
Bit 2 = 1	1	0
Bit 2 = 0	0	0

Example

The following procedure gets information from the Windows function GetVersion(), which returns the major and minor DOS and Windows versions as a CLONG. The "High Byte of the High Word" contains the major DOS version, the "Low Byte of the High Word" contains the minor DOS version. The "High Byte of the Low Word" specifies the minor version of Windows and the "Low byte of the Low Word" specifies the major Windows version. The Bit functions BITAND() and BITRSHIFT() are used to extract the proper values.

```
FUNCTION GVer
PARAMETER VerType
#define HiWord(x) (BITRSHIFT(BITAND;
    (x,4294901760),16))
#define LoWord(x) (BITAND(x,65535))
#define HiByte(x) (LTRIM(STR(BITRSHIFT(BITAND;
    (x,65280), 8))))
```

BITLSHIFT()

```
#define LoByte(x) (LTRIM(STR(BITAND(x,255))))
EXTERN CLONG GetVersion()  krnl386.exe
z=GetVersion()
IF UPPER(verType)="DOS"
    RETURN HiByte(HiWord(z))+ "." +LoByte(HiWord(z))
ENDIF
IF LEFT(UPPER(verType),3)="WIN"
    RETURN LoByte(LoWord(z))+ "." +HiByte(LoWord(z))
ENDIF
```

Portability

Not supported in dBASE IV or dBASE III PLUS.

See Also

BITLSHIFT(), BITOR(), BITRSHIFT(), BITSET(), BITXOR(), EXTERN, HTOI(), ITOH()

BITLSHIFT()

Windows programming

Returns a number generated by shifting another number's bits to the left.

Syntax

BITLSHIFT(<expN1>, <expN2>)

Description

BITLSHIFT() is used by advanced programmers to make binary interpretations of numeric values. These values are often returned by functions prototyped with EXTERN to access resources in Windows API or other DLL files. Interpreting such values often requires analysis and manipulation of individual bits.

BITLSHIFT() moves each bit in the numeric value <expN1> to the left the number of times you specify in <expN2>. Each time the bits are shifted, the least significant digit (the bit farthest to the right) is set to 0, and the most significant digit (the bit farthest to the left) is lost.

Example

The following example uses BITLSHIFT() to move the binary value of the decimal number 15 (00001111 binary) 4 bits left, resulting in the binary number 11110000, which equals 240 decimal:

```
? BITLSHIFT(15,4)      && Returns 240
```

Portability

Not supported in dBASE IV or dBASE III PLUS.

See Also

BITAND(), BITOR(), BITRSHIFT(), BITSET(), BITXOR(), EXTERN, HTOI(), ITOH()

BITOR()

Performs a bitwise OR operation on two numeric values and returns the result.

Syntax

BITOR(<expN1>, <expN2>)

Description

BITOR() is used by advanced programmers to make binary interpretations of numeric values. These values are often returned by functions prototyped with EXTERN to access resources in Windows API or other DLL files. Interpreting such values often requires analysis and manipulation of individual bits.

BITOR() compares bits in the numeric value <expN1> with corresponding bits in the numeric value <expN2>. When either or both bits in the same position are on (set to 1), the corresponding bit in the returned value is on. When neither element is on, the bit is off (set to 0).

	Bit 1 = 1	Bit 1 = 0
Bit 2 = 1	1	1
Bit 2 = 0	1	0

Example

The following example uses BITOR() to make bit comparisons of the binary equivalents of the numbers 64 (1000000 binary) and 48 (110000 binary). The result of 112 has a binary value of 1110000:

```
? BITOR(64,48)           && Returns decimal 112
```

Portability

Not supported in dBASE IV or dBASE III PLUS.

See Also

BITAND(), BITLSHIFT(), BITRSHIFT(), BITSET(), BITXOR(), EXTERN, HTOI(), ITOH()

BITRSHIFT()

Returns a number generated by shifting another number's bits to the right.

Syntax

BITRSHIFT(<expN1>, <expN2>)

Description

BITRSHIFT() is used by advanced programmers to make binary interpretations of numeric values. These values are often returned by functions prototyped with EXTERN

BITSET()

to access resources in Windows API or other DLL files. Interpreting such values often requires analysis and manipulation of individual bits.

BITRSHIFT() moves each bit in the numeric value *<expN1>* to the right the number of times you specify in *<expN2>*. Each time the bits are shifted, the most significant digit (the bit farthest to the left) is set to 0, and the least significant digit (the bit farthest to the right) is lost.

Example

The following example uses BITRSHIFT() to move the binary value of the decimal number 80 (1010000 binary) 4 bits right, resulting in the binary number 00000101, which equals 5 decimal:

```
? BITRSHIFT(80,4)      && Returns 5
```

For another example of using BITRSHIFT(), see BITAND().

Portability

Not supported in dBASE IV or dBASE III PLUS.

See Also

BITAND(), BITLSHIFT(), BITOR(), BITSET(), BITXOR(), EXTERN, HTOI(), ITOH()

BITSET()

Windows programming

Returns .T. if a specified bit in a numeric value is on.

Syntax

BITSET (*<expN1>*, *<expN2>*)

<expN1> A numeric value to evaluate.

<expN2> A decimal integer from 0 to 31 representing a position starting from the right end of *<expN1>*. For example, the binary representation of 3 is 00000011; bit number 0 is on, bit number 2 is off.

Description

BITSET() is used by advanced programmers to make binary interpretations of numeric values. These values are often returned by functions prototyped with EXTERN to access resources in Windows API or other DLL files. Interpreting such values often requires analysis and manipulation of individual bits.

BITSET() evaluates the number *<expN1>* and returns .T. if the bit in position *<expN2>* is on (set to 1), or .F. if it is off (set to 0).

Example

The following examples use BITSET() to return a setting of On (.T.) or Off (.F.) for various binary bits of the number 229 (11100101 binary):

```
mNum=229
? BITSET(mNum,0)      && Returns .T.
? BITSET(mNum,1)      && Returns .F.
? BITSET(mNum,6)      && Returns .T.
```

Portability
Not supported in dBASE IV or dBASE III PLUS.

See Also
BITAND(), BITLSHIFT(), BITOR(), BITRSHIFT(), BITXOR(), EXTERN

BITXOR()

Windows programming

Performs a bitwise exclusive OR operation on two numeric values and returns the result.

Syntax
BITXOR(<expN1>, <expN2>)

Description
BITXOR() is used by advanced programmers to make binary interpretations of numeric values. These values are often returned by functions prototyped with EXTERN to access resources in Windows API or other DLL files. Interpreting such values often requires analysis and manipulation of individual bits.

BITXOR() compares bits in a numeric value <expN1> with corresponding bits in the numeric value <expN2>. When one (and only one) of two bits in the same position are on (set to 1), the corresponding bit in the returned value is on. In any other case, the bit is off (set to 0).

	Bit 1 = 1	Bit 1 = 0
Bit 2 = 1	0	1
Bit 2 = 0	1	0

This operation is known as exclusive OR, since one bit (and only one bit) must be set on for the corresponding bit in the returned value to be set on.

Example
The following example uses BITXOR() to compare the binary bit makeup of 90 (01011010) to 214 (11010110) and return the number 140 (10001100):

```
mNum1=90
mNum2=214
? BITXOR(mNum1,mNum2)      && Returns 140
```

Portability
Not supported in dBASE IV or dBASE III PLUS.

See Also
BITAND(), BITLSHIFT(), BITOR(), BITSET(), BITRSHIFT(), HTOI(), ITOH()

BLANK

Fields and records

Fills fields in records with blanks.

Syntax
BLANK
[<scope>
[FOR <condition 1>
[WHILE <condition 2>
[FIELDS
 <field list> | [LIKE <skeleton 1>] [EXCEPT <skeleton 2>]]
[REINDEX]

<scope> The number of records to blank. RECORD <n> identifies a single record by its record number. NEXT <n> identifies *n* records, beginning with the current record. ALL specifies all records. REST specifies all records from the current record to the end of the file.

FOR <condition 1>
WHILE <condition 2> Determines which records are affected by BLANK. FOR restricts BLANK to records that meet <condition 1>. WHILE starts with the current record and continues with each subsequent record as long as <condition 2> is true.

FIELDS <field list> | LIKE <skeleton 1> | EXCEPT <skeleton 2> The fields to blank. Without FIELDS, BLANK replaces all field values. If you specify FIELDS LIKE <skeleton 1>, the BLANK command restricts the fields that dBASE makes blank to the fields that match <skeleton 1>. Conversely, if you specify FIELDS EXCEPT <skeleton 2>, the BLANK command makes all fields blank except those whose names match <skeleton 2>.

REINDEX Rebuilds all open indexes after BLANK finishes executing. Without REINDEX, BLANK updates all open indexes after each record is made blank.

Description
Use BLANK to replace fields in the current table with blanks. EMPTY() and ISBLANK() return .T. for a field whose value has been replaced using BLANK. BLANK fills an existing record with the same values as APPEND BLANK. Updates to open indexes are performed after each record or a set of records is blanked.

The following table shows the blank values that are displayed for fields of each data type:

Data type	Displayed value
Character	Spaces
Date	" / / "
Float	Empty (nonzero) value
Logical	.F.

Data type	Displayed value
Numeric	Empty (nonzero) value
Binary, Memo, and OLE	No text or graphic

Use BLANK to replace logical, float, and numeric fields with true blank values, which REPLACE can't do. When you use REPLACE to replace a numeric or float field with 0, dBASE treats it as a zero value rather than a blank value. The distinction between empty and zero values in numeric and float fields can be significant when you use commands such as AVERAGE and CALCULATE.

Example

The following example uses BLANK to reuse records that have been marked for deletion in the natural order file, by blanking the record, removing the deletion mark and entering EDIT:

```
USE Clients
? Reuse()
EDIT

FUNCTION Reuse
SET DELETED OFF
LOCATE FOR DELETED()
IF .NOT. FOUND()
  APPEND BLANK
ELSE
  BLANK          && blanks field values
  RecNum=RECNO()
  RECALL RECNO()
  GOTO RecNum
ENDIF
RETURN .T.
```

Portability

Not supported in dBASE III PLUS.

See Also

APPEND, ISBLANK(), EMPTY(), REPLACE

BOF()

Fields and records

Indicates if the record pointer in a table is at the beginning of the file.

Syntax

BOF([<alias>])

<alias> A work area number (1 through 225), letter (A through J), or alias name. The work area letter or alias name must be enclosed in quotes.

Description

BOF() returns .T. when the record pointer is before the first logical record of the table in the specified work area; otherwise, it returns .F. For example, if you issue SKIP -1 when the record pointer is on the first record, BOF() returns .T.

If no table is open in the specified work area, BOF() also returns .F.

Example

The following example defines a form and two pushbuttons for moving the record pointer. BOF() and EOF() are used to avoid a BOF() or EOF() error alert at either end of the table:

```
SET PROCEDURE TO PROGRAM(1) ADDITIVE
USE Clients
DEFINE FORM F1
DEFINE PUSHBUTTON Pb1 OF F1 AT 10,10;
    PROPERTY Text "Previous", Width 8, OnClick Back
DEFINE PUSHBUTTON Pb2 OF F1 AT 10,22;
    PROPERTY Text "Next", Width 8, OnClick Forward
OPEN FORM F1

FUNCTION Back
IF .NOT. BOF()
    SKIP-1
ENDIF
RETURN .T.

FUNCTION Forward
IF .NOT. EOF()
    SKIP
ENDIF
RETURN .T.
```

See Also

EOF(), RECNO(), SKIP

BOOKMARK()

Fields and records

Returns a bookmark for the current record. Bookmarks are used in place of record number for tables that don't support record pointers (for example, Paradox and SQL tables).

Syntax

BOOKMARK()

Description

BOOKMARK() returns a value for the current record in tables that don't support record pointers. When used with the GO command, bookmarks let you navigate to particular marked records.

The value returned by `BOOKMARK()` is of a special unprintable data type called bookmark. `BOOKMARK()` returns an empty bookmark if no table is open in the current work area.

Bookmark values can be used in all commands and functions that can otherwise use a record number, and with all the relational operators, `=`, `<`, `<=`, `>`, and `>=`, for comparisons.

Example

The following example lets the user press *F9* while in a browse to mark a record to edit after continuing to browse and leaving the Table Editor window:

```
SET DBTYPE TO PARADOX
SET TALK OFF
PUBLIC Srch1
USE Customer
ON KEY LABEL F9 DO GoBack
BROWSE
GOTO Srch1
EDIT
RETURN

PROCEDURE GoBack
Srch1=BOOKMARK()
RETURN
```

Portability

Not supported in dBASE IV or in dBASE III PLUS.

See Also

`GO`, `RECNO()`

BROWSE

Fields and records

Provides display and editing of records in a table format.

Syntax

```
BROWSE
[<scope>]
[FOR <condition 1>]
[WHILE <condition 2>]
[COLOR [<standard text>]
[, [<enhanced text>]
[, [<perimeter color>]
[, [<background color>]]]]]
[FIELDS <field 1> [<field option list 1>] |
<calculated field 1> = <exp 1> [<calculated field option list 1>]
[, <field 2> [<field option list 2>] |
<calculated field 2> = <exp 2> [<calculated field option list 2>]...]]
[FORMAT]
```

```
[FREEZE <field 3>]
[KEY <exp 3> [, <exp 4>] [EXCLUDE]]
[LOCK <expN 2>]
[NOAPPEND]
[NODELETE]
[NOEDIT | NOMODIFY]
[NOFOLLOW]
[NOINIT]
[NOORGANIZE]
[NORMAL]
[NOTOGGLE]
[NOWAIT]
[TITLE <expC 1>]
[WIDTH <expN 3>]
[WINDOW <window name>]
```

<scope> The number of records to browse. ALL specifies all records. REST specifies all records from the current record to the end of the file.

FOR <condition 1>

WHILE <condition 2> Determines which records are affected by BROWSE. FOR restricts BROWSE to records that meet <condition 1>. WHILE starts processing with the current record and continues with each subsequent record as long as <condition 2> is true.

COLOR [<standard text>] [, [<enhanced text>] [, [<perimeter color>] [, [<background color>]]]

Specifies the colors of standard text, enhanced text, and the perimeter of the Table Records window. To specify the colors of these elements separately, use the <standard text>, <enhanced text>, and <perimeter> options. You can also use the <background> option if you have a monitor with a uniform background.

<standard text>	Color attributes of command messages and screen output. For example, the output of the ? and @ ... SAY commands appear in standard text.
<enhanced text>	Color attributes of enhanced text areas, such as @...GET fields and highlighted BROWSE data cells.
<perimeter>	Color attributes of the perimeter that borders the area displaying text onscreen.
<background>	Color attributes of the background for display systems (such as monochrome) with uniform backgrounds. <background> includes two parameters: a background color and an attribute.

The <standard text> and <enhanced text> attributes include three settings: a foreground color, a background color, and an optional color for creating blended (hatched) backgrounds. Separate each setting with a forward slash (/).

For more information about color settings, see SET COLOR TO and SET COLOR OF.

FIELDS <field 1> [<field option list 1>] |

<calculated field 1> = <exp 1> [<calculated field option list 1>]

[, <field 2> [<field option list 2>] |

<calculated field 2> = <exp 2> [<calculated field option list 2>] ...]] Displays the specified fields, in the order they're listed, in the Table Records window. Options for <field option list 1>, <field option list 2>, which apply to <field 1>, <field 2>, and so on, affect the way these fields are displayed. These options are as follows:

\<column width>	The width of the column within which <field 1> appears when <field 1> is character type
\B = <exp 1>, <exp 2> [\F]	RANGE option; forces any value entered in <field 1> to fall within <exp 1> and <exp 2>, inclusive. RANGE REQUIRED option; the \F option prevents the cursor from leaving <field 1> and the editing session from ending until the value falls between <exp 1> and <exp 2>, inclusive
\C=<color>	COLOR option; sets the foreground and/or background colors of the column according to the values specified in <color>
\H = <expC>	HEADER option; causes <expC> to appear above the field column in the Table Records window, replacing the field name
\P = <expC>	PICTURE option; displays <field 1> according to the PICTURE or FUNCTION clause <expC>
\R	READ-ONLY option; specifies that <field 1> is read-only and can't be edited
\V = <condition> [\F] [\E = <expC>]	VALID option; allows a new <field 1> value to be entered only when <condition> evaluates to .T. VALID REQUIRED option; the \F option prevents the cursor from leaving <field 1> and the editing session from ending until <condition> evaluates to .T. ERROR MESSAGE option; \E = <expC> causes <expC> to appear when <condition> evaluates to .F.
\W = <condition>	WHEN option; allows <field 1> to be edited only when <condition> evaluates to .T.

Note You may also use the "/" character when specifying only a single option in a field option list.

Read-only calculated fields are composed of an assigned field name and an expression that results in the calculated field value, as with *Commission = Rate * Saleprice*. Options

for calculated fields affect the way these fields are displayed. These options are as follows:

<code>\<column width></code>	The width of the column within which <i><calculated field 1></i> is displayed
<code>\H = <expC></code>	Causes <i><expC></i> to appear above the calculated field column in the Table Records window, replacing the calculated field name

FORMAT Causes BROWSE to accept and display input according to the specifications of a format file opened with SET FORMAT. Data entered must conform to any PICTURE, FUNCTION, RANGE, and VALID clauses in the format file.

FREEZE <field 3> Restricts editing to *<field 3>*, although other fields are visible.

KEY <exp 3> [, <exp 4>] [EXCLUDE] When the table has a master index, displays records whose key field value matches or comes after *<exp 3>*, or falls between *<exp 3>* and *<exp 4>*. EXCLUDE specifies that the range of values is non-inclusive.

LOCK <expN 2> Keeps the first *<expN 2>* fields in place onscreen as you move the cursor to fields on the right.

NOAPPEND Prevents you from adding records from the Table Records window.

NODELETE Prevents the marking of records for deletion from within the Table Records window.

NOEDIT | NOMODIFY Prevents you from modifying records from the Table Records window.

NOFOLLOW When the current table has a master index, causes the cursor to remain in place when you change the key field in a record, rather than follow the record to its new location in the indexing order. Otherwise, the record pointer follows the record to its new location.

NOINIT Causes BROWSE to execute the options specified with the previous BROWSE command. Use NOINIT if a program calls BROWSE several times or if you issue BROWSE several times from the Command window, and you want the same options. Specify the command options the first time you use BROWSE, and issue BROWSE NOINIT for subsequent use in the same session.

NOORGANIZE Disables options to index, sort, and remove records.

NORMAL When BROWSE is issued from an active window, displays the Table Records window in normal, full-screen mode with default or defined colors set, ignoring the defined colors of the window. When you leave BROWSE, Visual dBASE returns you to the active window. Without NORMAL, the table records appears in the active window.

NOTOGGLE Prevents toggling from browse mode to edit mode.

NOWAIT Continues execution of a program after a Table Records window is displayed; otherwise, program execution is suspended until after the Table Records window is closed.

TITLE <expC 1> Causes *<expC 1>* to appear as the title of the Table Records window.

WIDTH <expN 3> Specifies the display width for character fields in the Table Records window. If a field is wider than the specified width, you can scroll the field within the specified width. The <expN 3> argument must evaluate to a positive number.

WINDOW <window name> Activates the window <window name> and displays the table records in the window.

Description

The BROWSE command provides an interactive window-oriented environment for displaying and editing more than one record at a time. Use SET RELATION command to view fields from records in linked tables. While BROWSE is in effect and unless you specify NOTOGGLE, you can press *F2* or choose View | Edit to toggle to edit mode, to display a single record.

To move between records in the BROWSE display, you can press the Up and Down arrows to move one record at a time, and *PgUp* and *PgDn* to move one window frame at a time. You can also use the window controls and the mouse, and choose from various menu options to control operations performed with BROWSE. See the *User's Guide* for more information on editing data and navigating in the Table Records window.

When you're through editing a table, press *Ctrl+W* to exit and save changes to the current record or choose the File | Save and Close option. To exit without saving changes to the current record, press *Ctrl+Q*, choose File | Abandon and Close, or double-click the Control menu. If you're using BROWSE or EDIT in a program, exiting returns program control to the command line immediately following the BROWSE or EDIT command line.

Example

The following example uses BROWSE to view selected fields from two related tables, to add custom field headers and to specify read-only fields:

```
USE Contact ORDER CompCode IN SELECT()
USE Company IN SELECT()
SELECT Company
SET RELATION TO CompCode INTO Contact
BROWSE FIELDS ;
    Contact->CompCode /R /H="Company Code", ;
    Company->Company /R, ;
    Contact->Contact /R /H="Contact Person", ;
    Company->Street1 /R, Company->Street2 /R, ;
    Company->City /R, Company->State_Prov /R
CLOSE ALL
```

See Also

APPEND, EDIT, SET FIELDS, SET FORMAT, SET MEMOWIDTH, SET RELATION.
SET WINDOW OF MEMO

Links object code files (.PRO, .WFO) and resources into a Windows executable file (.EXE) if the optional *Visual* dBASE Compiler is installed.

Syntax

```
BUILD <filename list> | FROM <response filename>  
[ICON <icon filename>]  
[SPLASH <bitmap format filename>]  
[TO <executable filename>]  
<filename list>
```

List of filenames, separated by commas, to be linked into the executable. Unless otherwise specified, the filename extensions are assumed to be .PRO.

FROM <response filename> Build the executable from the list of files listed in <response filename>. Unless otherwise specified, the extension of the response file is assumed to be .RSP. See the online help for details on the format of the response file.

ICON <icon filename> The optional icon (.ICO) file which is displayed when the executable is minimized. The icon file is also the default icon when the executable is represented in Program Manager.

```
SPLASH <bmp format filename>
```

The optional graphics (.BMP) file that displays as the executable is loading.

TO <executable filename> The name of the Windows executable file (.EXE) produced by BUILD. If not specified, the executable filename defaults to the name of the first file listed in the <filename list> or the <response file>.

Description

Use the BUILD command to link previously compiled dBASE program (.PRO, .WFO) files, Windows resource files (such as .BMP and .ICO files), and other files needed to support an application into a Windows executable (.EXE) file. See the online help for the Visual dBASE compiler for details on creating executable files from dBASE programs.

Portability

Not supported in dBASE IV or dBASE III PLUS. Very similar in functionality to the BDL.EXE linker utility in the dBASE Compiler for DOS.

See Also

COMPILE, DO, SET FORMAT, SET PROCEDURE

CALCULATE

Table organization

Performs financial and statistical operations for values of records in the current table.

C

Syntax

```
CALCULATE <function list>
[<scope>]
[FOR <condition 1>]
[WHILE <condition 2>]
[TO <memvar list> | TO ARRAY <array name>]
```

<function list> You can use one or more of the following functions:

Function	Purpose	Return value
AVG(<expN>)	Calculates the average of the specified numeric or float expression.	CALCULATE AVG() returns a float value.
CNT()	Counts the number of records of the current table.	CALCULATE CNT() returns a numeric value.
MAX(<expC> <expN> <expD>)	Calculates the maximum value of the specified numeric, float, character, or date expression.	CALCULATE MAX() returns the same data type as the expression specified.
MIN(<expC> <expN> <expD>)	Calculates the minimum value of the specified numeric, float, character, or date expression.	CALCULATE MIN() returns the same data type as the expression specified.
NPV(<expN 1>, <expN 2> [, <expN 3>])	Calculates the net present value of the numeric or float values in <expN 2>; <expN 1> is the periodic interest rate, expressed as a decimal; <expN 3> is the initial investment and is generally a negative number.	CALCULATE NPV() returns a float value.
STD(<expN>)	Calculates the standard deviation of the specified numeric or float expression.	CALCULATE STD() returns a float value.
SUM(<expN>)	Calculates the sum of the specified numeric or float expression.	CALCULATE SUM() returns a float value.
VAR(<expN>)	Calculates the variance of the specified numeric field.	CALCULATE VAR() returns a float value.

<scope> The number of records to calculate. RECORD <n> identifies a single record by its record number. NEXT <n> identifies *n* records, beginning with the current record. ALL specifies all records. REST specifies all records from the current record to the end of the file.

FOR <condition 1>

WHILE <condition 2> Determines which records are affected by CALCULATE. FOR restricts CALCULATE to records that meet <condition 1>, starting with the first record of the table or view and continuing until reaching the end of file. WHILE starts processing with the current record and continues with each subsequent record as long as <condition 2> is true.

TO <memvar list> | TO ARRAY <array name> The TO <memvar list> option initializes the memory variables in <memvar list> and stores the results of CALCULATE in the variables. The TO ARRAY <array name> stores the results of CALCULATE in the existing array <array name>.

Description

CALCULATE uses one or more of the eight associated functions listed in the previous table to calculate and store sums, maximums, minimums, averages, variances, standard deviations, or net present values of specified expressions of the current table.

CALCULATE can also return the count or number of records in the current table. These special functions (with the exception of the MAX() function) can be used only with CALCULATE.

If SET TALK is ON, CALCULATE displays the results in the result pane.

CALCULATE can use the same function on different expressions or different functions on the same expression. For instance, if your table contains a Salary field and a Bonus field, you can issue the command, CALCULATE SUM(Salary), SUM(Bonus), AVG(Salary), AVG(12*(Salary + Bonus)).

You can calculate values in a work area other than the current work area if you set a relation between the work areas.

CALCULATE stores results to memory variables or to an existing array in the order of the listed functions. If you store the results to memory variables, specify the same number of memory variables as the number of functions in the CALCULATE command line.

When you use CALCULATE TO ARRAY <array name>, Visual dBASE stores each result to one array element. CALCULATE can store results to a multidimensional array. For example, if you use DECLARE test[3,2], CALCULATE can store values returned by up to six functions in test[1,1], test[1,2], test[2,1], test[2,2], test[3,1], test[3,2], in that order.

You can store to an array without the ARRAY keyword if you explicitly reference the array subscript(s). For example, the command line CALCULATE AVG(Salary), MAX(Salary) TO test[2], test[3] stores the average and maximum of the salary field to two array elements.

If you use CALCULATE...TO ARRAY, declare an array with at least as many elements as there are functions in the command line. Do not include the subscripts of the array in the CALCULATE command line. For instance, if the current table contains a numeric field, Numfield, and you want to store its minimum and maximum to an array *minmax*, you can issue the command DECLARE minmax[2] and then specify CALCULATE MIN(Numfield), MAX(Numfield) TO ARRAY minmax.

CALCULATE treats a blank numeric or float field as containing 0 and includes the field in its calculations. For example, if you calculate the average of a numeric field in a table containing ten records, five of which are blank, CALCULATE divides the sum by 10 to find the average. Furthermore, if you calculate the minimum of the same table field and five records contain numeric data and the five others are blank in the same fields, CALCULATE returns 0 as the minimum. If you want to exclude blank fields when using CALCULATE, be sure to specify a condition such as FOR .NOT. ISBLANK(numfield).

Although you can use the SUM or AVERAGE commands to find sums and averages, CALCULATE is faster because it runs through the table just once while making all specified calculations.

Example

The following example uses CALCULATE to place results in memory variables:

```
USE Company
PUBLIC AV,CN,MX,MN,VR,ST
CALCULATE AVG(Ytd_Sales),CNT(),MAX(Ytd_sales),;
    MIN(Ytd_sales),SUM(Ytd_sales),VAR(Ytd_sales),;
    STD(Ytd_sales);
    TO AV,CN,MX,MN,VR,ST
? AV,CN,MX,MN,VR,ST
WAIT
* Display Average, Record Count, Maximum,
* Minimum, Sum, Variance and Standard Deviation
```

The following example uses CALCULATE to place the results in an array:

```
USE Company
PUBLIC ARRAY Results[10]
CALCULATE AVG(Ytd_Sales),CNT(),MAX(Ytd_sales),;
    MIN(Ytd_sales),SUM(Ytd_sales),VAR(Ytd_sales),;
    STD(Ytd_sales);
    TO ARRAY Results
FOR i=1 TO 7
    ? i,Results[i]
NEXT i
* Display the results
```

Portability

Not supported in dBASE III PLUS.

See Also

AVERAGE, DECLARE, MAX(), MIN(), SET FIELDS, SET RELATION, SUM

CANCEL

Programs

Halts program execution, closes program files, clears private memory variables, and returns control to the Command window.

Syntax

CANCEL

Description

Use CANCEL to cancel program execution. When dBASE encounters CANCEL in a program file, it ignores any text on lines following CANCEL, halting program execution. You can also issue CANCEL in the Command window when a program is suspended (such as with SUSPEND) to cancel execution of the suspended program. When dBASE halts program execution, it clears all private memory variables for that program and returns control to the Command window.

CATALOG()

Example

The following example uses CANCEL to halt program execution when the Exit pushbutton within a form is clicked with the mouse:

```
DEFINE FORM Main FROM 2,2 TO 20,30;
PROPERTY MDI .F.
DEFINE PUSHBUTTON Exit OF Main AT 13,10;
PROPERTY;
TEXT "Exit",;
OnClick {;Cancel}
READMODAL(Main.Exit)
```

See Also

DO, QUIT, RESUME, RETRY, RETURN, SUSPEND

CATALOG()

Table basics

Returns the name of the current catalog file.

Syntax

CATALOG()

Description

CATALOG() returns the name of the current catalog opened with the SET CATALOG TO command or the Navigator. If no catalog is currently open, CATALOG() returns an empty string ("").

Example

The following example uses CATALOG() to determine the name of the currently open catalog:

```
SET CATALOG TO Learn
catname = CATALOG()
USE (catname) IN SELECT() NOUPDATE AGAIN
```

Portability

Not supported in dBASE IV or in dBASE III PLUS.

See Also

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

Changes the current default drive or directory.

Syntax

CD

[<path>]

<path> A character expression indicating the default path. To specify a root path, start <path> with a backslash (\) or the root directory (as with C:\). If <path> doesn't begin with \, .. (two periods), or the root directory, dBASE begins the path with the current directory.

Description

CD works like the DOS commands CD and CHDIR. It lets you change the current working drive and directory without exiting to DOS. Use CD to change the current working directory to any valid drive and path. If you're unsure whether a drive is valid, use VALIDDRIVE() before issuing CD. The current directory appears in the File Viewer window.

CD .. (two periods) changes the directory to the directory one level above the current directory. CD without the option <path> displays the current drive and directory path.

Another way to access files on different directories is with the command SET PATH. You can specify one or more search paths, and dBASE uses these paths to locate files not on the current directory. Use SET PATH when an application's files are in several directories.

CD works like SET DIRECTORY, except SET DIRECTORY TO (with no argument) returns you to the default working directory, instead of displaying the current directory. See SET DIRECTORY for more information.

Example

The following example saves the current directory, then changes directory several times, and finally returns to the original directory:

```
* This example assumes that
* directories were created by:
MD D:\Project
MD D:\Project\Programs
MD D:\Project\Data
MD D:\Project\Backup
CD D:\Project\Programs
MD C:\Editor

Olddir=SET("DIRECTORY")
? SET("DIRECTORY")      && D:\Project\PROGRAMS
CD ..
? SET("DIRECTORY")      && D:\Project
CD Data
? SET("DIRECTORY")      && D:\Project\Data
CD ..\BACKUP
```

CDOW()

```
? SET("DIRECTORY")      && D:\Project\BACKUP
CD C:\Editor
? SET("DIRECTORY")      && C: changes to C:\Editor
CD &olddir
? SET("DIRECTORY")      && D:\Project\Programs
```

Portability

Not supported in dBASE IV or dBASE III PLUS, but SET DIRECTORY is supported in dBASE IV.

See Also

MKDIR, SET DEFAULT, SET DIRECTORY, SET PATH, VALIDDRIVE()

CDOW()

Date and time data

Returns the day of the week corresponding to a specified date expression as a character string.

Syntax

CDOW(<expD>)

<expD> The date expression whose corresponding weekday name to return.

Description

CDOW() returns a character string containing the name of the day of the week on which a date falls. To return the day of the week as a number from 1 to 7, use DOW().

Enter <expD> in the current date format as determined by SET DATE, DBASEWIN.INI, or the International option of the Windows Control Panel (in that order). That is, SET DATE settings override those in DBASEWIN.INI, and settings in DBASEWIN.INI override those in the Windows Control Panel. Be sure <expD> matches the date format in use when your program runs.

If you pass an invalid date to CDOW(), dBASE converts the date to a valid one and returns the weekday name of that date. If you pass an empty or non-date expression delimited with braces ({ }) to CDOW(), it returns "Unknown". If you pass a non-date expression or an expression that isn't delimited with braces to CDOW(), it returns an error.

Example

The following example uses CDOW() to return the day of the week for a date type memory variable.

```
CLEAR
SET TALK OFF
Turncen={12/31/1999}
? CENTER("Welcome in the new century on")
? CENTER(CDOW(Turncen)+ " night,")
? CENTER(CMONTH(Turncen)+ " " +;
```

```
LTRIM(STR(DAY(Turncen)))+ "st " +;
LTRIM(STR(YEAR(Turncen)))
```

This code returns:

```
*           Welcome in the new century on
*           Friday night,
*           December 31st 1999
```

CDOW() is useful for filtering or manipulating data by the day of the week. For example, if you wanted to determine the number of transactions that occurred on weekends, the following example would apply.

```
USE Flights
SET EXACT OFF
COUNT FOR CDOW(Date) = "S" TO Weekend
? Weekend           && Returns the number of weekend flights
```

To list all Wednesday flights from the Flights table.

```
SET FILTER TO CDOW(Date)="Wednesday"
LIST OFF           && "OFF" suppresses record numbers
```

See Also

CMONTH(), DATE(), DAY(), DOW(), SET CENTURY, SET DATE, YEAR()

CEILING()

Numeric data

Returns the nearest integer that is greater than or equal to a specified number.

Syntax

CEILING(<expN>)

<expN> A numeric or float number, from which to determine and return the integer that is greater than or equal to it.

Description

CEILING() returns the nearest integer that is greater than or equal to <expN>. If you pass a number with any digits other than 0 as decimal digits, CEILING() returns the nearest integer that is greater than the number. If you pass an integer to CEILING(), or a number with only 0s for decimal digits, it returns the integer equal to the number.

For example, if the default number of decimal places is 2,

- CEILING(2.10) returns 3.00
- CEILING(-2.10) returns -2.00
- CEILING(2.00) returns 2.00
- CEILING(2) returns 2
- CEILING(-2.00) returns -2.00

See the table in the description of INT() that compares INT(), FLOOR(), CEILING(), and ROUND().

The value returned by CEILING() has the same data type as *<expN>*.

Example

The following example uses CEILING() to return the passed value, rounded up to the next whole number:

```
SET TALK OFF
msrp = 79.95
cost = 45.50
percent = cost/msrp * 100
? "The cost is " + STR(CEILING(percent),2,0) + "% of the retail price."
SET TALK ON
```

See INT() for another example of CEILING().

Portability

Not supported in dBASE III PLUS. In dBASE IV, CEILING() doesn't display any decimal places, regardless of the value of SET DECIMALS.

See Also

FLOOR(), INT(), ROUND()

CENTER()

String data

Returns a character string that contains a string centered in a line of specified length.

Syntax

CENTER(*<expC>* | *<memo field>* [, *<length expN>* [, *<pad expC>*]])

<expC> | *<memo field>* The text to center.

<length expN> The length of the resulting line of text, which must be less than 32766, the maximum length of a string. If *<length expN>* isn't specified, CENTER() assumes a line length of 80 characters.

<pad expC> The single character to pad the string with if *<length expN>* is greater than the number of characters in *<expC>* | *<memo field>*. If *<length expN>* is equal to or less than the number of characters in *<expC>* | *<memo field>*, *<pad expC>* is ignored.

If *<pad expC>* is more than one character, CENTER() uses only the first character. If *<pad expC>* isn't specified, CENTER() pads with spaces.

Description

CENTER() returns a character expression or a memo field with the requisite number of leading and trailing spaces to center it in a line that is a specified number of characters wide. If you specify a memo field, CENTER() centers the full text of the field, not each line of text.

If you specify a padding character, CENTER() pads either side of the character expression or memo field with that character (or with the first character, if you specify more than one) rather than with spaces.

To create the resulting string, CENTER() performs the following steps.

- Subtracts the length of *<expC>* or *<memo field>* from *<length expN>*
- Divides the result in half and rounds up if necessary
- Pads *<expC>* or *<memo field>* on either side with that number of spaces or the first character in *<pad expC>*

If the length of *<expC>* or *<memo field>* is greater than *<length expN>*, CENTER() does the following:

- Subtracts *<length expN>* from the length of *<expC>* or *<memo field>*
- Divides the result in half and rounds up if necessary
- Truncates both sides of *<expC>* or *<memo field>* by that many characters

When the result of subtracting the length of *<expC>* or *<memo field>* from *<length expN>* is an odd number, CENTER() pads one less space on the left if the difference is positive, or truncates one less character on the left if the difference is negative.

Example

The following example uses CENTER() to position header text displayed above a listing of the Company table. Nested counting loops display 8 records at a time with header information in the results pane of the Command window:

```
SET TALK OFF
USE Clients
CLEAR
DO WHILE .NOT. EOF()
    ? CENTER("Clients Database Report",80)
    ?
    ? CENTER("Run on " + CDOY(DATE()) + ;
        " " + DTOC(DATE()),80,"-")
    ? "Company" AT 2,"Phone" AT 40,"Balance Date" AT 60
    ? "*****" AT 2,"*****" AT 40,"*****" AT 60
    Cnt=1
    DO WHILE Cnt<9 .AND. .NOT. EOF()
        ? Company AT 2, Phone AT 40, CDOY(Baldate)+",";
        + DTOC(Baldate) AT 60
        SKIP
        Cnt=cnt+1
    ENDDO
    ?
    WAIT
    CLEAR
    ENDDO
CLOSE ALL
```

Portability

Not supported in dBASE IV or dBASE III PLUS.

CERROR()

See Also

LEN(), REPLICATE(), SET MEMOWIDTH, TRANSFORM()

CERROR()

Error handling and debugging

Returns the number of the last compiler error.

Syntax

CERROR()

Description

Use CERROR() before executing a new program to test whether the source code compiles successfully. If no compiler error occurs, CERROR() returns 0. CERROR() is updated each time you or dBASE compile a program or format file. CERROR() isn't affected by warning messages generated by compiling.

Use CERROR() in a program file. If you issue ? CERROR() in the Command window, it returns 0. (This is because dBASE is compiling the "? CERROR()" command itself, which does not cause a compiler error.)

See the table in the description of ERROR() that compares ERROR(), MESSAGE(), DBERROR(), DBMESSAGE(), SQLERROR(), SQLMESSAGE(), and CERROR().

See online Help for a listing of all error messages.

Example

The following program segment uses CERROR() in a DO WHILE loop to make the user edit the program until it compiles successfully:

```
DO WHILE .T.
  Clear
  MODIFY COMMAND USER.PRG
  ON ERROR ? ERROR( ), MESSAGE( ), CERROR( )
  COMPILE USER.PRG
  IF CERROR()>0
    ?
    WAIT "Your program didn't compile. Press a key to edit your .PRG."
  LOOP
ELSE
  EXIT
ENDIF
ENDDO
```

Portability

Not supported in dBASE III PLUS.

See Also

COMPILE, DBERROR(), DBMESSAGE(), ERROR(), MESSAGE(), ON ERROR

CHANGE

Fields and records

C

Provides display and editing of data in the current table, one record at a time.

Syntax

```
CHANGE
[<starting record expN 1> | <bookmark>]
[<scope>]
[FOR <condition 1>]
[WHILE <condition 2>]
[COLOR [<standard text>]
  [, [<enhanced text>]
    [, [<perimeter color>]
      [, <background color>]]]]
[COLUMNAR]
[COMPRESS]
[FIELDS <field 1> [<field option list 1>] |
  <calculated field 1> = <exp 1> [<calculated field option list 1>]
  [, <field 2> [<field option list 2>] |
    <calculated field 2> = <exp 2> [<calculated field option list 2>]...]]
[FORMAT]
[FREEZE <field 3>]
[KEY <exp 3> [, <exp 4>]]
[LOCK <expN 2>]
[NOAPPEND]
[NODELETE]
[NOEDIT | NOMODIFY]
[NOFOLLOW]
[NOINIT]
[NORMAL]
[NOTOGGLE]
[NOWAIT]
[TITLE <expC 1>]
[WIDTH <expN 3>]
[WINDOW <window name>]
```

Description

CHANGE and EDIT are equivalent commands. See EDIT for a complete discussion of these commands. CHANGE without the FIELDS option displays all fields of the current record.

Example

See EDIT for an example of CHANGE, substituting CHANGE for EDIT in the example.

See Also

BROWSE, EDIT

CHANGE()

Shared data

Returns .T. if another user has changed a record since it was read from a table file.

Syntax

CHANGE([<alias>])

<alias> A work area number (1 through 225), letter (A through J), or alias name. The work area letter or alias name must be enclosed in quotes. If you don't include <alias>, CHANGE() returns information on the current table.

Description

Use CHANGE() to determine if another user has made changes to a record since it was read from the table file. If the record has been changed, you might want to display a message to the user before allowing the user to continue.

For CHANGE() to return information, the current or specified alias table must have a _dbaseLock field. Use CONVERT to add a _dbaseLock field to a table. If the table doesn't contain a _dbaseLock field, CHANGE() returns .F.

CHANGE() compares the counter in the workstation's memory image of _dbaseLock to the counter stored on disk. If they are different, the record has changed, and CHANGE() returns .T.

You can reset the value of CHANGE() to .F. by moving the record pointer. GOTO RECNO() rereads the current record's _dbaseLock field, and a subsequent CHANGE() returns .F., unless another user has changed the record in the interim between moving to it and issuing CHANGE().

Note CHANGE() doesn't test SQL databases or Paradox tables.

Example

The following example opens the Company table, takes the user to a selected record, and opens a form to review the year to date data. Within the form, ReviewYtd, the user can reset that value. (The ReviewYtd form definition is not shown). On leaving the form, the program uses CHANGE() to determine if another user has changed the current record. If not, REPLACE updates the value of the YTD_Sales field:

```
USE COMPANY
SEEK mCompany
OPEN FORM ReviewYtd
IF .NOT. CHANGE()
* Has another user changed this record?
  REPLACE YTD_Sales WITH mYSales
ELSE
  GOTO RECNO()      && Re-read this record
ENDIF
```

Portability

Not supported in dBASE III PLUS.

See Also

CONVERT, FLOCK(), LKSYS(), RLOCK(), SET EXCLUSIVE, SET REFRESH

C

CHARSET()

Environment

Returns the name of the character set the current table or a specified table is using. If no table is open and you issue CHARSET() without an argument, it returns the global character set in use.

Syntax

CHARSET([<*alias*>])

<*alias*> A work area number (1 through 225), letter (A through J), or alias name. The work area letter or alias name must be enclosed in quotes.

Description

Use CHARSET() to learn which character set the current table or a specified table is using. If you don't pass CHARSET() an argument, it returns the name of the character set of the current table or, if no tables are open, the global character set in use. CHARSET() also returns information on Paradox and SQL databases.

The character set a table's data is stored in depends on the language driver setting that was in effect when the table was created. With *Visual* dBASE, you can choose the language driver that applies to your dBASE data in the [CommandSettings] section in the DBASEWIN.INI file. For more information on character sets and language drivers, see PG_CHARLANG.

The value CHRSET() returns is a subset of the value LDRIVER() returns. For more information, see LDRIVER().

Example

This example shows the CHARSET() function and a sample response:

```
? CHARSET( )           && Returns DOS:437
```

Portability

Not supported in dBASE IV or dBASE III PLUS.

See Also

ANSI(), LDRIVER(), OEM(), SET LDCHECK

CHOOSEPRINTER()

Printing

Opens the Print Setup dialog box for choosing a printer or specifying print options, and resets the appropriate system memory variables for the printer or print options you specify. Returns true (.T.) if you exit the dialog box by choosing OK, and false (.F.) if you choose Cancel.

Syntax

CHOOSEPRINTER(<title expC>)

<title expC> Specifies a title to display in the Print Setup dialog box.

Description

Use CHOOSEPRINTER() to select a new printer or change printer options using the Print Setup dialog box. You can also open the Print Setup dialog box by choosing File | Print Setup. In the Print Setup dialog box, you can select a new installed printer or select options such as paper size, paper source, and orientation (portrait or landscape).

CHOOSEPRINTER() modifies the value of the SET PRINTER TO setting, and of the _pdriver, _plength, and _porientation memory variables.

For a printer to appear in the Specific Printer list, you must have installed it previously through the Windows Print Manager. You can also change printer drivers in Windows with the Printers program of the Windows Control Panel.

To activate a specific printer driver, you can also use _pdriver.

Example

CHOOSEPRINTER() opens the print setup dialog box. Newprinter returns true if the printer was reset:

```
Newprinter=CHOOSEPRINTER()
Newprinter=CHOOSEPRINTER("Choose a dot matrix"+ " printer")
```

Portability

Not supported in dBASE III PLUS or dBASE IV.

See Also

_pdriver, CLOSE..., SET DEVICE, SET PRINTER

CHR()

Expressions and type conversion

Returns the character equivalent of a specified ASCII value.

Syntax

CHR(<expN>)

<expN> The numeric ASCII value, from 0 to 255, inclusive, whose character equivalent to return.

Description

CHR() is the inverse function of ASC(). CHR() accepts an ASCII value and returns its character, while ASC() accepts a character and returns its ASCII value.

You can use CHR() with ASC() to convert an ASCII value to its character equivalent. See the ASCII table in Appendix E for a listing of ASCII values and their corresponding characters.

CHR(7) returns a tone, or beep. Use SET BELL TO [*<frequency expN>*, *<duration expN>*] to set the frequency and duration of the tone CHR(7) returns.

Example

The following example uses CHR() to display a special character on screen. Special characters are not available on the keyboard but can be printed or displayed with the CHR() function (example assumes default OEM character set).

```
? "The total is 50" + CHR(241) + "2"
* The ASCII value 241 displays a plus/minus symbol
```

Printer control codes can be sent to a printer with the ?? command and CHR() as follows.

```
SET PRINT ON
?? CHR(27) + "E"
* Resets an HP LaserJet to default mode by
* sending ESCAPE (CHR(27)) plus a control code.
* Could also be sent ?? CHR(27) + CHR(69).
```

See Also

ANSI(), ASC(), OEM(), SET BELL

CLASS...ENDCLASS

Objects

Declares a custom class and specifies the member variables and functions for that class.

Syntax

```
CLASS <class name> [(<parameters>)] [CUSTOM]
[PARAMETERS <parameters>]
[FROM <filename>]
[OF <superclass name> [(<parameters>)]]
[PROTECT <propertyList>]
[<constructor code>]
[<member functions>]
ENDCLASS
```

CUSTOM Specifies that the new object is a custom control. For information on custom controls, see Chapter 15 of the *Programmer's Guide*.

<class name> The name you give to the new class.

OF <superclass name> Specifies that the class you create inherits the properties and methods of *<superclass name>*. For example, you can give your new class all the properties and methods of the listbox class or another class you create with CLASS...ENDCLASS.

FROM <filename> *<filename>* specifies the file containing the definition code for the *<superclass>*, if the *<superclass>* is not defined in the same file as the class.

PROTECT <propertyList> <propertyList> is a list of properties and/or methods of the class which are to be accessible only by other members of the class, and by classes derived from the class.

<constructor code> Code that is executed when you create an object of class <class name>. Constructor code includes all commands between the CLASS and ENDCLASS keywords except code in <member functions>.

<member functions> Procedures and functions that you declare between the CLASS and ENDCLASS keywords. These subroutines make up the methods of the new class.

Description

Use CLASS...ENDCLASS to create a new class.

A class is a specification, or template, for a type of object. *Visual* dBASE provides many standard classes, such as Form and Entryfield; for example, when you create a form, you are creating a new form object that has the standard properties and methods from the Form class. However, when you declare a custom class with CLASS...ENDCLASS, *you* specify the properties and methods that objects derived from the new class will have.

You create properties for the new class with <constructor code>. Constructor code executes when you create an object of the class. Although constructor code can contain any dBASE commands, it usually contains only property and method assignment statements.

Properties and methods can be protected to prevent the user of the class from reading or changing the protected property values, or calling the protected methods from outside of the class.

When you create a new property in a class declaration, preface the property name with the *This* keyword. *This* references the object you create. For example, the following code sample includes a class declaration. The declaration uses *This* to specify that TagName, a new property, is a member of the new class TableFile.

```
xFile = NEW TableFile()
? xFile.TagName
? xFile.FileNameId()

CLASS TableFile
  This.TagName = "XORDER"
  FUNCTION FileNameId&& Custom method.
  RETURN DBF()
ENDCLASS
```

You create custom methods for the class with <member functions>, which can consist of procedure declarations or user-defined function declarations. FUNCTION FileNameId is an example of a custom method.

Example

The following example uses CLASS...ENDCLASS to define a class of objects within a form that displays pictures from the Pictures table in the SAMPLES directory. This example is an extract from PICTURES.WFM in the SAMPLES directory:

```

** END HEADER -- do not remove this line*
* Generated on 06/27/94
*
LOCAL f
f = NEW PFORM()
f.Open()

CLASS PFORM OF FORM
PROTECT HelpFile, HelpId
    this.HelpFile = ""
    this.Width = 97.00
    this.Maximize = .F.
    this.Minimize = .F.
    this.Height = 24.82
    this.Left = 19.40
    this.Text = "Pictures Form"
    this.Top = 0.53
    this.ColorNormal = "BG/B"
    this.OnOpen = {;create session}
    this.View = "PICTURES.QBE"
    this.HelpId = ""

    DEFINE PUSHBUTTON SOUND OF THIS;
        PROPERTY;
            Width 18.00;;
            Default .T.,;
            OnClick {;play sound binary pictures->sound},;
            Height 3.00;;
            Left 4.00;;
            Text "Sound",;
            Top 8.00;;
            ColorNormal "N/W",;
            FontName "Courier",;
            FontSize 16.00

    DEFINE LISTBOX THINGS OF THIS;
        PROPERTY;
            Width 18.40;;
            DataSource "FIELD NAME",;
            ColorHighLight "W+/B",;
            Height 5.47;;
            Left 3.60;;
            Top 13.41;;
            ColorNormal "bg+/b",;
            FontName "Fixedsys",;
            FontSize 11.25;;
            ID 800

    DEFINE IMAGE PICTURE OF THIS;
        PROPERTY;
            Width 62.00;;
            DataSource "BINARY PICTURES->BITMAPOLE",;
            Height 18.00;;
            Left 25.00;;
            Top 5.00;;
            ID 88

```

CLEAR

```
DEFINE TEXT TITLE OF THIS;
  PROPERTY;
    Width      70.00;;
    Height     4.30;;
    Left       20.00;;
    Text "Sights and Sounds",;
    Top        0.00;;
    ColorNormal "gr+/b",;
    FontName "Serif",;
    Border .F.,;
    FontSize   32.00
* Provide methods to get and set the HelpFile
* and HelpID properties, since the user can't
* access them directly
FUNCTION GetHelpFile
RETURN This.HelpFile
FUNCTION GetHelpID
RETURN This.HelpID
FUNCTION SetHelpFile(cHelpFile)
  IF TYPE("cHelpFile") = "C"
...   This.HelpFile = cHelpFile
  ENDIF
RETURN This.HelpFile
FUNCTION SetHelpID(cHelpID)
  IF TYPE("cHelpID") = "C"
    This.HelpID = cHelpID
  ENDIF
RETURN This.HelpID

ENDCLASS

PROCEDURE Sound_OnClick
PLAY SOUND Binary Pictures->Sound

PROCEDURE ClosePictures
USE IN Pictures
FORM.CLOSE()
```

Portability

Not supported in dBASE IV or dBASE III PLUS.

See Also

DEFINE, REDEFINE

CLEAR

Environment

Erases the contents of the results pane of the Command window or the current dBASE IV window, and clears pending @...GETs.

Syntax

CLEAR
[CHARACTER <expC>]

CHARACTER <expC> Fills the results pane of the Command window or the current dBASE IV window with the first character of the expression <expC>. If you do not specify a character, CLEAR fills the window with the last character specified by <expC>, if any, or else with space characters.

Description

Use CLEAR to remove the contents of the results pane of the Command window or the current dBASE IV window. Use the option CHARACTER <expC> with CLEAR to specify a character for dBASE to repeat when clearing the window. To clear only *parts* of the window, use @...CLEAR.

If you use CLEAR without an option after issuing CLEAR CHARACTER <expC>, CLEAR uses <expC> again. To reset CLEAR, issue CLEAR CHARACTER " " (with a space character or empty string as <expC>).

If you don't want to close and remove open GETs from the display, issue READ SAVE before CLEAR.

Portability

Not supported in dBASE III PLUS.

See Also

CLEAR PROGRAM, CLOSE..., RELEASE, SET PROCEDURE, SET LIBRARY

CLEAR AUTOMEM

Fields and records

Initializes automem variables with empty values for the current table.

Syntax

CLEAR AUTOMEM

Description

Use CLEAR AUTOMEM to initialize a set of automem variables containing empty values for the current table. CLEAR AUTOMEM creates any automem variables that don't exist already. If the variables exist, CLEAR AUTOMEM reinitializes them. If no table is in use, CLEAR AUTOMEM doesn't create any variables.

Automem variables have the same names and data types as the fields in an active table. You can create empty automem variables automatically for the current table by using CLEAR AUTOMEM or USE...AUTOMEM, or manually by using STORE.

Use CLEAR AUTOMEM if automem variables are used more than once within a program, so that automem variables are reset with empty values and don't carry over into a subsequent form display or record. For example, use CLEAR AUTOMEM within a loop that adds data by means of a form, automem variables, and APPEND

CLEAR AUTOMEM

AUTOMEM, so that the values entered on one pass of the loop don't appear in the entry form on the next pass.

You can also use CLEAR AUTOMEM to create automem variables if you didn't create them with USE...AUTOMEM. For example, you can create automem variables from the Command window for a table already in use.

Example

The following example uses CLEAR AUTOMEM to enable the user to edit AUTOMEM variables for a new record. A new record will be added only if the user confirms that the data held in automem variables is correct:

```
SET TALK OFF
CLEAR
USE Clients
AddMoreData()
RETURN

FUNCTION AddMoreData
@0,0 to 8, 70
@10,20 to 12,45
CLEAR AUTOMEM
DO WHILE .T.
    lConfirm = .f.
    @11,22 CLEAR TO 11,39
    @1,1 SAY 'ID' GET m->CLIENT_ID
    @1,10 SAY 'COMPANY' GET m->COMPANY
    @3,1 SAY 'Contact' GET m->CONTACT
    @4,1 SAY 'Address' GET m->ADDRESS
    @6,1 SAY 'City' GET m->CITY
    @6,23 SAY 'State/Province' GET STATE_PROV
    @6,54 SAY 'Zip' GET ZIP_P_CODE
    READ
    IF READKEY() = 12
        EXIT
    ELSE
        @ 11,22 SAY "Data Correct, Y-N?";
        GET lConfirm picture 'Y'
        READ
        IF lConfirm
            APPEND AUTOMEM
            CLEAR AUTOMEM
        ENDIF
    ENDIF
ENDDO
RETURN .T.
```

See Also

APPEND, STORE, USE

CLEAR FIELDS

Fields and records

Removes the fields list defined with the SET FIELDS TO command.

C

Syntax

CLEAR FIELDS

Description

Use CLEAR FIELDS to remove the SET FIELDS TO *<field list>* setting in all work areas and automatically turn SET FIELDS to OFF, thus making all fields in all open tables accessible. You can use CLEAR FIELDS prior to specifying a new fields list with SET FIELDS TO. You might also want to use CLEAR FIELDS at the end of a program. CLEAR FIELDS has the same effect as SET FIELDS TO with no options.

Example

See SET FIELDS for examples of CLEAR FIELDS.

See Also

SET FIELDS

CLEAR GETS

Input/Output

Clears all current @...GET fields. This command is supported primarily for compatibility with dBASE IV. In *Visual dBASE*, use DEFINE with the Text and EntryField classes for displaying and accepting information on a form.

For complete syntax information on CLEAR GETS, see online Help. For more information about working with *Visual dBASE* forms, see the Forms chapters in the *User's Guide*.

CLEAR MEMORY

Memory variables

Clears all user-defined memory variables.

Syntax

CLEAR MEMORY

Description

Use CLEAR MEMORY to release all memory variables (except system memory variables) in all currently active sessions, including those declared PUBLIC and STATIC and those initialized in higher-level subroutines. CLEAR MEMORY has no effect on system memory variables.

Note CLEAR MEMORY does not normally release objects. However, if the only reference to an object is in a memory variable, CLEAR MEMORY releases the object.

CLEAR MEMORY, whether issued in a program or in the Command window, has the same effect as issuing RELEASE ALL in the Command window. However, CLEAR MEMORY in a program is different from RELEASE ALL in a program. RELEASE ALL in a program clears only memory variables created at the same program level as the RELEASE ALL statement. Unlike CLEAR MEMORY, it has no effect on higher-level, public, or static variables.

CLEAR MEMORY is also different from RELEASE AUTOMEM. In a program or in the Command window, RELEASE AUTOMEM erases only automem variables associated with the current table. (For information about AUTOMEM variables, see USE, CLEAR AUTOMEM, and RELEASE AUTOMEM.)

To clear only selected memory variables, use RELEASE.

Example

The following example uses CLEAR MEMORY to release variables held in an array after they have been saved to a .DBF file used for backup purposes:

```
SET SAFETY OFF
USE Clients EXCLUSIVE
INDEX ON Client_ID TAG Client_ID
DECLARE StbyInfo[RECCOUNT(),FLDCOUNT()]
COPY TO ARRAY StbyInfo
*
* Perform various file operations...
*
COPY STRUCTURE TO Backup.DBF WITH PRODUCTION
USE Backup
APPEND FROM ARRAY StbyInfo
CLEAR MEMORY      && Releases values held in StbyInfo array
* Subsequent operations that require additional memory...
CLOSE ALL
```

See Also

CLEAR AUTOMEM, PRIVATE, PUBLIC, RELEASE, RELEASE AUTOMEM, STATIC, USE

CLEAR MENUS

dBASE IV menus

Clears all dBASE IV bar menus from the screen and their definitions from memory. This command is supported primarily for compatibility with dBASE IV. In *Visual dBASE*, use RELEASE OBJECT to clear an object from a form.

For complete syntax information on CLEAR MENUS, see online Help. For information about defining forms, see the Forms chapters in the *User's Guide*.

CLEAR POPUPS

dBASE IV menus

C

Clears all dBASE IV pop-up menus from the screen and their definitions from memory. This command is supported primarily for compatibility with dBASE IV. In *Visual* dBASE, use `RELEASE OBJECT` to clear an object from a form.

For complete syntax information on `CLEAR POPUPS`, see online Help. For information about defining forms, see the Forms chapters in the *User's Guide*.

CLEAR PROGRAM

Programs

Clears from memory all compiled program files that aren't currently executing and aren't currently open with `SET FORMAT`, `SET PROCEDURE`, or `SET LIBRARY`.

Syntax

`CLEAR PROGRAM`

Description

When you run a program with `DO`, dBASE loads into memory the compiled program and all compiled programs, procedures, user-defined functions (UDFs), and format files the program calls with `DO`, `SET FORMAT`, `SET PROCEDURE`, and `SET LIBRARY`. These files remain in memory until you issue `CLEAR PROGRAM` or until dBASE needs more memory—for example, when you open a large table. dBASE's internal dynamic memory management can also make more memory available by clearing from memory any programs that haven't been active for a while.

`CLEAR PROGRAM` clears all inactive compiled programs from memory. The command doesn't clear programs that are currently executing or files the current program opens with `SET FORMAT`, `SET PROCEDURE` or `SET LIBRARY`. However, if you close a called file (for example, with `CLOSE PROCEDURE` or `CLOSE FORMAT`), a subsequent `CLEAR PROGRAM` clears the closed file from memory.

Use `CLEAR PROGRAM` to clear memory before executing tasks that require a lot of memory. For example, use `CLEAR PROGRAM` before you run a memory-intensive DOS command, before using an external text editor, or before you open a large number of tables.

Don't use `CLEAR PROGRAM` frequently, because dBASE keeps programs in memory to speed execution. Repeatedly loading called files into memory slows program execution.

Example

The following example uses `CLEAR PROGRAM` to release memory occupied by unused programs, procedures, or format files:

```
** Main.PRG **
CLEAR
DO Set_Main           && External PRG
DO Get_Mov            && Internal Proc
```

CLEAR SCREENS

```
CLEAR PROGRAM                && Release memory;  
                               by clearing programs  
  
DO Mov_Rpt  
  
PROCEDURE Get_Mov  
USE Movies  
INDEX ON Director TAG Mov_Dir  
GOTO rec_no  
DO WHILE Director = Mov_Dir  
    ? Title  
    SKIP  
ENDDO WHILE Director = Mov_Dir  
GOTO rec_no  
CLOSE DATABASES  
RETURN
```

Portability

Not supported in dBASE IV or dBASE III PLUS.

See Also

DO, CLEAR ALL, CLOSE..., SET FORMAT, SET LIBRARY, SET PROCEDURE

CLEAR SCREENS

Input/Output

Removes from memory all variables created by SAVE SCREEN, and clears the Command window buffer. This command is supported primarily for compatibility with dBASE IV.

For complete syntax information on CLEAR SCREENS, see online Help.

CLEAR TYPEAHEAD

Keyboard and mouse events

Clears the typeahead buffer, where keystrokes are stored while dBASE is busy processing other data. This command is supported primarily for compatibility with dBASE IV. In *Visual* dBASE, forms do not use the typeahead buffer.

For complete syntax information on CLEAR TYPEAHEAD, see online Help. For information about working with forms, see the Forms chapters in the *User's Guide*.

CLEAR WINDOWS

dBASE IV windows

Clears all dBASE IV-style windows from the screen and removes their definitions from memory. This command is supported primarily for compatibility with dBASE IV. In *Visual* dBASE, use CLOSE FORMS or RELEASE OBJECT to close or release a form.

For complete syntax information on CLEAR WINDOWS, see online Help. For information about defining forms, see the Forms chapters in the *User's Guide*.

CLOSE...

C

Closes various file types.

CLOSE ALL

Table basics

CLOSE ALTERNATE

Input/Output

CLOSE DATABASES [*<database name list>*]

Table basics

CLOSE FORMAT

Input/Output

CLOSE FORMS [*<form name list>*]

Forms

CLOSE INDEXES

Table organization

CLOSE PRINTER

Printing

CLOSE PROCEDURE *<filename>*

Programs

CLOSE TABLES

Table basics

CLOSE ALL Closes tables, databases, and files of all type (including low-level files) and reselects work area 1. However, files opened by SET DEVICE TO *<filename>* or SET PRINTER TO *<filename>* and SQL databases remain open. To close SQL databases, use CLOSE DATABASES.

CLOSE ALTERNATE Closes text (.TXT) files opened with the SET ALTERNATE command.

CLOSE DATABASES [*<database name list>*] Closes a specified list of databases (separated by commas) or all open databases, including all tables, and associated index (.MDX and .NDX) files, memo (.DBT) files, and format (.FMT) files for each database.

CLOSE FORMAT Closes format (.FMT) files open in the current work area.

CLOSE FORMS [*<form name list>*] Closes the specified forms and executes the standard close routines for the forms and the objects that are contained in them.

CLOSE INDEXES Closes index (.MDX and .NDX) files open in the current work area. This option does not close the production .MDX file.

CLOSE PRINTER Closes a file opened with the SET PRINTER command.

CLOSE PROCEDURE *<filename>* Closes an open procedure file.

CLOSE TABLES Closes all tables in all work areas or all tables in the current database, if one is selected.

Description

If you use the CREATE SESSION command, the CLOSE commands affect only the files in the current *session*. A session is similar to a user session in a multi-user environment—each session manages its own set of work areas, and a file opened as exclusive cannot be accessed by other sessions.

By default, the *Visual* dBASE interactive environment operates with CREATE SESSION. For example, if you open tables through the user interface (for example, by double-clicking the file icon), dBASE opens each table in its own session (observe the Command window and you'll notice that a CREATE SESSION command is executed each time you open a table). If you then type CLOSE TABLES in the Command window, the tables remain open because they are protected by their sessions. As in a multi-user environment, a user can't close another user's open files.

The Command window itself runs in an independent session; that's another reason why the CLOSE commands issued there have no effect on files opened through the user interface. If, however, you open tables by using the USE and BROWSE commands in the Command window, those tables are in the Command window session. Therefore, when you type CLOSE TABLES in the Command window, those tables are closed.

For more information about sessions, see CREATE SESSION.

Example

The following example uses CLOSE ALL to close all open tables upon completion of a report:

```
SET SAFETY OFF
SET TALK OFF
USE Contact EXCLUSIVE IN SELECT()
INDEX ON COMPCODE TAG COMPCODE
USE Company IN SELECT()
SELECT Company
SET RELATION TO CompCode INTO Contact
GO TOP
DO ListComp
CLOSE ALL

PROCEDURE ListComp
SET ALTERNATE TO Complist
SET ALTERNATE ON
CLEAR
? "Company" AT 5, "Contact Person" AT 45
?
DO WHILE .NOT. EOF()
    ? Company->Company AT 5, Contact->Contact AT 45
    ? "In " + REPLICATE("-", 7) + "->" AT 5,;
        TRIM(Company->City) + ", " +;
        TRIM(Company->State_Prov) AT 25
    SKIP
ENDDO
RETURN
```

See Also

CLEAR ALL, CREATE SESSION

CMONTH()

Date and time data

C

Returns the name of the month in which a specified date expression falls.

Syntax

CMONTH(<expD>)

<expD> The date expression, in the current date format, whose corresponding month name to return.

Description

CMONTH() returns a character string containing the name of the month in which a date falls.

Enter <expD> in the current date format as determined by SET DATE, DBASEWIN.INI, or the International option of the Windows Control Panel (in that order). That is, SET DATE settings override those in DBASEWIN.INI, and settings in DBASEWIN.INI override those in the Windows Control Panel. Be sure <expD> matches the date format in use when your program runs.

If you pass an invalid date to CMONTH(), dBASE converts the date to a valid one and returns the month name of that date. If you pass an empty or non-date expression delimited with braces ({ }) to CMONTH(), it returns "Unknown". If you pass a non-date expression or an expression that isn't delimited with braces to CMONTH(), it returns an error.

Example

The following example uses CMONTH(), DAY(), and YEAR() to return the literal day of week, month, date and year in a character string.

```
SET TALK OFF
SET CENTURY ON
date = {04/01/94}
? FullDate(date)      && Function is called returns "Friday, April 1, 1994"

FUNCTION FullDate
PARAMETERS date
full_date = CDOW(date) + ", " + CMONTH(date) + ;
" " + LTRIM(STR(DAY(date))) + ;
", " + LTRIM(STR(YEAR(date)))
RETURN full_date
```

See Also

CDOW(), DAY(), MONTH(), SET DATE, YEAR()

COL()

Input/Output

Returns the number of the current column position in the results pane of the Command window or the current dBASE IV window. This command is supported primarily for

compatibility with dBASE IV. In *Visual* dBASE, use the Left property of a class to determine its horizontal position on a form.

For complete syntax information on COL(), see online Help. For more information about working with *Visual* dBASE forms, see the Forms chapters in the *User's Guide*.

COMMIT()

Shared data

Ends a transaction initiated by BEGINTRANS() and writes to the open files any changes made during the transaction. Returns .T. if the data was committed successfully.

Syntax

COMMIT([<database name expC>])

<database name expC> The name of the database in which to complete the transaction.

- If you began the transaction with BEGINTRANS(<database name expC>), you must issue COMMIT(<database name expC>). If instead you issue COMMIT(), dBASE ignores the COMMIT() statement.
- If you began the transaction with BEGINTRANS(), <database name expC> is an optional COMMIT() argument. If you include it, it must refer to the same database as the SET DATABASE TO statement that preceded BEGINTRANS().

Description

Use COMMIT() to end the open transaction and write changes to any open files. To end a transaction without writing changes to the file, use ROLLBACK(). For more information on transactions, see BEGINTRANS().

Example

The following example begins a transaction with BEGINTRANS(). It opens a multi-user version of Company.dbf and attempts to set all values in the Ytd_Sales field to 0. ON ERROR detects any error which might occur. In particular, it will detect if another user has locked any record in Company.dbf. If an error occurs, ROLLBACK() resets all values. Otherwise COMMIT() writes the changes to disk:

```
CLOSE ALL
SET PROCEDURE TO PROGRAM(1) ADDITIVE
SET EXCLUSIVE OFF

BEGINTRANS()

TransErr=.f.
ON ERROR DO TransErr          && Activates ON ERROR trap

USE L:\MultiUse\Company
REPLACE ALL Ytd_Sales WITH 0
ON ERROR                      && Disables ON ERROR

IF TransErr
    ? "Rollback"
```



```

        ROLLBACK()                && Restore data
    ELSE
        ? "Commit"
        COMMIT()                  && Save changes
    ENDIF

PROC TransErr
WAIT "Warning: Transaction Fails"
TransErr=.t.

```

Portability

Not supported in dBASE IV or dBASE III PLUS. COMMIT() replaces the COMMIT command in dBASE IV.

See Also

BEGINTRANS(), ROLLBACK(), SET EXCLUSIVE

COMPILE

Programs

Compiles program files (.PRG, .WFM), creating object code files (.PRO, .WFO).

Syntax

```

COMPILE <filename> <filename skeleton>
[AUTO]
[LOG <filename>]
[TO <response filename>]

```

<filename> <filename skeleton> The file to compile. The *<filename skeleton>* options display a dialog box from which you can select a file. If you specify a file without including its path, dBASE looks for the file in the current directory. If you specify a file without including its extension, dBASE assumes .PRG.

AUTO The optional AUTO clause causes the compiler to detect automatically which files are called by your program, and to compile those that have changed since the last compile. All .PRG and associated files must be in the current directory to use this option.

LOG <filename> LOG causes the compiler to write any error or warning messages to <filename>.

TO <response filename> The TO clause causes the compiler to create a response file containing the names of the files output by the compiler (the object code files). If a specified source file cannot be successfully compiled, its name is included in the response file marked with an asterisk. The response file can be used by the BUILD command in the Visual dBASE compiler to link the object code files into an executable file.

Description

Use COMPILE to create compiled program files without executing or opening the files, or to compile only certain files. You can't run a program until it's been compiled.

Because a compiled file can't be read or modified, compiling a program protects your source code from modification by users of the program. By default, dBASE creates compiled object files in the same directory as the source code files.

COMPILE has several advantages over compiling files with DO, SET PROCEDURE, or SET FORMAT:

- COMPILE doesn't execute or open the specified files.
- If you write an application that contains many program files, you can use COMPILE to compile only those program files you change rather than all the program files of the application. To specify a date and time range for the programs to be compiled, use FDATE() and FTIME().
- COMPILE *<filename skeleton>* lets you compile groups of unrelated or related files.

When you compile a program, dBASE detects any syntax errors in the source file and displays an error message corresponding to the error in a dialog box that contains three buttons:

- *Cancel* cancels compilation (equivalent to pressing *Esc*).
- *Ignore* cancels compilation of the program containing the syntax error but continues compilation of the rest of the files that match *<filename skeleton>* if you specified a skeleton.
- *Fix* lets you fix the error by opening the source code in an editing window, positioning the insertion point at the point where the error occurred.

See the on-line help for information about compiling dBASE programs into standalone executable files.

Portability

By default, both dBASE IV and dBASE III PLUS create compiled object files in the current directory rather than in the same directory as the source code files.

See Also

CLEAR PROGRAM, DO, FDATE(), FTIME(), SET COVERAGE, SET DEVELOPMENT, SET FORMAT, SET PROCEDURE

CONTINUE

Table organization

Continues a search for the next record that meets the conditions specified in a previously issued LOCATE command.

Syntax

CONTINUE

Description

CONTINUE continues the search of the last LOCATE issued in the selected work area. After you issue the LOCATE command, the current table is searched sequentially for the

first record that matches the search criteria. Unless a WHILE *<condition>* is included, or *<scope>* is NEXT or REST, the search begins with the first record.

If a record is found, dBASE moves the record pointer to the matching record and displays the message **Record = <n>** (if SET TALK is ON). To continue the search, issue the CONTINUE command. If another match is found, CONTINUE displays the message **Record = <n>**, and FOUND() returns .T. If another match is not found, it displays the message **End of LOCATE scope** and positions the record pointer at the last record of the LOCATE scope or at end-of -file. Also, FOUND() returns .F.

If you issue CONTINUE without first issuing a LOCATE command for the current table, Visual dBASE returns an error message.

Example

The following example uses CONTINUE to find the next Company that is not in California. It relies on the LOCATE command:

```
USE Company
LOCATE FOR Company <> "CA"
* After LOCATE, CONTINUE can be used
IF EOF()
* IF .NOT. FOUND() can also be used to test LOCATE/CONTINUE
? "No companies found"
ENDIF
DO WHILE .NOT. EOF()
? Company, State_Prov
CONTINUE
ENDDO
```

See Also

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

CONVERT

Shared data

Adds a _dbaselock field to a table for storing multiuser lock information.

Syntax

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

TO <expN> Specifies the length of the multiuser information field to add to the current table. The <expN> argument can be a number from 8 to 24, inclusive. The default is 16.

Description

Use CONVERT to add a character field, _dbaselock, to the structure of the current table. Use the option TO <expN> to specify the length of the field. If you issue CONVERT without the TO <expN> option, the width of the field is 16. If you want to change the length of the _dbaselock field after using CONVERT, you can issue CONVERT again on the same table. To view the contents of the _dbaselock field, use LKSYS().

Note You must use the table exclusively (USE...EXCLUSIVE) before issuing CONVERT. You must also turn SET DELETED OFF before issuing CONVERT or reindex the table (REINDEX) after issuing CONVERT.

When you issue CONVERT, dBASE copies the current table to a new file with extension .CVT, then creates a new .DBF file containing the _dbaseLock field. The .CVT table contains the original file structure before CONVERT.

The _dbaseLock field contains the following values:

Count	A 2-byte hexadecimal number used by CHANGE()
Time	A 3-byte hexadecimal number that records the time a lock was placed
Date	A 3-byte hexadecimal number that records the date a lock was placed
Name	A 0- to 16-character representation of the login name of the user who placed a lock, if a lock is active

The count, time, and date portions of the _dbaseLock field always make up its first 8 characters. If you accept the default 16-character width of the _dbaseLock field, the login name is truncated to 8 characters. If you set the field width to fewer than 16 characters, the login name is truncated the necessary amount. If you set the width of <expN> to 8 characters, the login name doesn't appear at all.

Every time a record is updated, dBASE rewrites the count portion of _dbaseLock. If you issue CHANGE(), dBASE reads the count portion from disk and compares it to the previous value it stored in memory when the record was initially read. If the values are different, another user has changed the record, and CHANGE() returns .T. For more information, see CHANGE().

LKSYS() returns the login name, date, and time portions of the _dbaseLock field. If you place a file lock on the table containing the _dbaseLock field, the value in the _dbaseLock field of the first record contains the information used by CHANGE() and LKSYS(). For more information, see LKSYS().

Note CONVERT doesn't affect SQL databases or Paradox tables.

Example

The following series of commands issued at the Command window add a field to the active table to track multiuser change information:

```
SET DELETED OFF
USE Company EXCLUSIVE
DISPLAY STRUCTURE      && Note structure
CONVERT TO 24
DISPLAY STRUCTURE      && Note added _dbaseLock field with 24 byte size
```

Portability

Not supported in dBASE III PLUS.

See Also

CHANGE(), FLOCK(), LKSYS(), LOCK(), NETWORK(), REINDEX, RLOCK(), SET DELETED, SET EXCLUSIVE, SET LOCK, SET REPROCESS, UNLOCK, USE

COPY

Table basics

C

Creates a new table and copies records to it from the current table. COPY also allows you to export data to non-dBASE files.

Syntax

```
COPY TO <filename> | ?
[<scope>]
[FOR <condition 1>]
[WHILE <condition 2>]
[FIELDS <field list>]
[[TYPE] SDF | DBMEMO3 | PARADOX | DBASE |
  DELIMITED [WITH
    <char> | BLANK]] |
[[WITH] PRODUCTION]
```

TO <filename> | ? Specifies the name of the table or file you want to create. COPY TO ? displays a dialog box, in which you can specify a new destination file. If you specify a file without including its path, *Visual dBASE* saves the file to the current drive and directory. If you specify a file without including an extension, defining a default table type with SET DBTYPE, or using one of the TYPE options, *Visual dBASE* assigns a .DBF extension.

You can also copy records to a table in a database (defined using the BDE Configuration Utility) by specifying the database as a prefix (enclosed in colons) to the name of the table, that is, *:database name:table name*. If the database is not already open, *Visual dBASE* displays a dialog box in which you specify the parameters, such as a login name and password, necessary to establish a connection to that database.

<scope> The number of records to copy to <filename>. RECORD <n> identifies a single record by its record number. NEXT <n> identifies *n* records, beginning with the current record. ALL specifies all records. REST specifies all records from the current record to the end of the file.

FOR <condition 1>

WHILE <condition 2> Determines which records are affected by COPY. FOR restricts COPY to records that meet <condition 1>. WHILE starts processing with the current record and continues with each subsequent record as long as <condition 2> is true.

FIELDS <field list> Specifies which fields to copy to the new table.

[[TYPE] SDF | DBMEMO3 | PARADOX | DBASE |

DELIMITED [WITH <char> | BLANK] Specifies the format of the file to which you want to copy data. The TYPE keyword is included for readability only; it has no effect on the operation of the command. The following table provides a description of the different file formats that are supported:

Type	Description
SDF	A System Data Format file. Records are fixed-length in the SDF file, and the end of a record is marked with a carriage return and a linefeed. If you don't include an extension for <filename>, <i>Visual dBASE</i> assigns a .TXT extension.
DBMEMO3	A table (.DBF) and memo (.DBT) files in dBASE III PLUS format.

Type	Description
PARADOX	A Paradox table. Records are Paradox rows, and fields are Paradox columns. <i>Visual</i> dBASE assigns a .DB extension.
DBASE	A dBASE table (the default). If you don't include an extension for <filename>, <i>Visual</i> dBASE assigns a .DBF extension.
DELIMITED	<p>A text in which</p> <ul style="list-style-type: none">• Character fields are delimited with quotation marks or with <char> if you use WITH <char>.• Logical fields are characters T or F.• Numeric fields are numbers.• Each carriage return and linefeed indicates a new record. <p>If you don't include an extension for <filename>, <i>Visual</i> dBASE assigns a .TXT extension.</p> <p>WITH <char></p> <p>Delimits character fields in a delimited text file with the character <char> instead of with quotation marks.</p> <p>WITH BLANK</p> <p>Separates delimited text file data with spaces instead of commas. Character fields aren't delimited with quotes or other delimiters.</p>

[WITH] PRODUCTION Specifies copying the production .MDX file along with the associated table. This option can be used only when copying to another dBASE table.

Description

Use COPY to copy all or part of a table to a file of the same or a different type. If an index is active, COPY arranges the records of the new table or file according to the indexed order. The COPY command does not copy a _dbaselock field in a table that you've created with CONVERT.

If you COPY a table containing a memo field to another dBASE table, *Visual* dBASE creates another file with the same name as the table but having a .DBT extension, and copies the contents of the memo field to it. If, however, you use the SDF or DELIMITED options and COPY to an ASCII text file, *Visual* dBASE doesn't copy the memo field .DBT file.

Deleted records are copied to the target file (if it's a dBASE table) unless a FOR or WHILE condition excludes them or unless SET DELETED is ON. Deleted records remain marked for deletion in the target dBASE table.

You can use COPY to create a file containing fields from more than one table. To do that, open the source tables in different work areas and define a relation between the tables. Use SET FIELDS TO to select the fields from each table that you want to copy to a new file. Before you issue the COPY command, SET FIELDS must be ON and you must be in the work area in which the parent table resides.

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

Example

The following example uses COPY to copy selected fields from two related tables to a new Paradox table:

```
SET SAFETY OFF
USE Contact Exclusive
INDEX ON CompCode TAG CompCode
USE Company IN SELECT()
SELECT Company
SET RELATION TO CompCode INTO Contact
COPY TO CntctLst FOR Company->State_Prov = "CA";
    FIELDS Company->Company,;
    Contact->CompCode, Contact->Contact, ;
    Company->Street1, Company->Street2, ;
    Company->City, Company->State_Prov,;
    Company->Zip_P_Code TYPE PARADOX
CLOSE DATABASES
SET DBTYPE TO PARADOX
USE CntctLst
BROWSE
CLOSE DATABASES
SET DBTYPE TO
```

C

See Also

APPEND FROM, CONVERT, COPY FILE, COPY STRUCTURE, COPY TABLE, COPY TO...STRUCTURE EXTENDED, IMPORT, SET DELETED, SET FIELDS

COPY BINARY

Fields and records

Copies the contents of the specified binary field to a file.

Syntax

```
COPY BINARY <field name> TO <filename>
[ADDITIVE]
```

<field name> Specifies the binary field to copy.

TO <filename> | ? Specifies the name of the file where the contents of the binary field are copied. For predefined binary file types, *Visual* dBASE assigns the appropriate extension, for example, .BMP, .WAV, and so on. For user-defined binary type fields, dBASE assigns a .TXT extension by default.

ADDITIVE Appends the contents of the binary field to the end of an existing file. Without the ADDITIVE option, *Visual* dBASE overwrites the previous contents of the file.

Description

Use COPY BINARY to export data from a binary field in the current record to a file. You can use binary fields to store text, images, sound, video, and other user-defined binary data.

If you specify the ADDITIVE option, *Visual* dBASE appends the contents of the binary field to the end of the named file, which lets you combine the contents of binary fields from more than one record. When you don't use ADDITIVE, *Visual* dBASE displays a warning message before overwriting an existing file if SET SAFETY is ON. Note that you can't combine the data from more than one field for many of the predefined binary data types. For example, you can store only a single image in a binary field or file, so do not use the ADDITIVE option of COPY BINARY when copying an image to a file.

Example

The following example uses COPY BINARY to copy bitmapped pictures and sound data stored in binary fields to external files. The newly created files are given a name comprised of the Name field contents, the digit 2 to distinguish the new file from old files and a .BMP or .WAV file extension:

```
USE Pictures
DO WHILE .NOT. EOF()
    bmp_file = TRIM(Name) + ".BMP"
    wav_file = TRIM(Name) + ".WAV"
    COPY BINARY Bitmap TO &bmp_file
    COPY BINARY Sound TO &wav_file
    SKIP
ENDDO
CLOSE ALL
```

Portability

Not supported in dBASE IV or dBASE III PLUS.

See Also

APPEND MEMO, BINTYPE(), CLASS IMAGE, COPY, COPY FILE, COPY MEMO, PLAY SOUND, REPLACE BINARY, RESTORE IMAGE

COPY FILE

Disk and file utilities

Duplicates a file.

Syntax

```
COPY FILE <filename 1> | ? | <filename skeleton 1>
TO <filename 2> | ? | <filename skeleton 2>
```

<filename 1> | ? | <filename skeleton 1> Identifies the file to duplicate (also known as the source file). ? and <filename skeleton> display a dialog box from which you can select a file to duplicate.

If you specify a source file without including its path, dBASE looks for the file in the current directory, then in the path(s) you specified with SET PATH. If you specify a source file without including its extension, dBASE assumes no extension.

TO <filename 2> | ? | <filename skeleton 2> Identifies the target file that will be created or overwritten by COPY FILE. The ? and <filename skeleton> options display a dialog box in which you specify the name of the target file and its directory.



Description

COPY FILE is a dBASE utility command that lets you duplicate an existing file at the operating system level. COPY FILE duplicates a single file of any type.

COPY FILE differs from the COPY command in DOS in that wildcards do not let you copy more than one file at a time. To copy more than one file at a time using wildcards, use !, RUN, or DOS, and execute the DOS COPY command.

If SET SAFETY is ON and a file exists with the same name as the target file, dBASE displays a dialog box asking if you want to overwrite the file. If SET SAFETY is OFF, any existing file with the same name is overwritten without warning.

COPY FILE does not automatically copy a .DBT or .MDX file associated with a .DBF file. If, for example, you copy a table file that has memo fields and you do not copy the associated .DBT file, an error message displays when you try to use the file. In such cases, it is best to use COPY instead of COPY FILE. Also, you must close a dBASE file before you can use COPY FILE on it.

Example

The following examples use COPY FILE:

```
COPY FILE Temp.prg TO B:Temp.prg
* Copies Temp.prg
COPY FILE *.DBF TO B:*.DBF
* Displays the open source dialog box
```

Portability

The ? and <filename skeleton> options are not supported in dBASE IV or dBASE III PLUS.

See Also

COPY, COPY INDEXES, COPY MEMO, COPY STRUCTURE, COPY TO ARRAY, COPY TO...STRUCTURE EXTENDED, RENAME, SET FULLPATH, SET PATH

COPY INDEXES

Table organization

Copies a list of .NDX files to index tags within a single .MDX file. (Applicable only to .NDX indexes created on dBASE tables.)

Syntax

```
COPY INDEXES <.ndx filename list>
[TO <.mdx filename> | ? ]
```

<.ndx filename list> Specifies a list of .NDX files (up to ten) that you want to create index tags for.

TO <.mdx filename> | ? Specifies the name of the .MDX file to add index tags to. By default, Visual dBASE assigns the index file the same name as the current table with an .MDX extension and saves the file in the current directory. The ? option displays a dialog box, in which you specify the name and location of the .MDX file.

Description

The COPY INDEXES command converts a list of .NDX files into index tags in a single .MDX file. If you do not specify an .MDX file with the TO *<mdx filename>* option, index tags are created in the production .MDX file, which has the same name as the table (and which is opened automatically when you open the associated table). If a production .MDX file does not exist, it is created with the same name as the table, and, for dBASE tables, the table header is updated to indicate the presence of the production index. If you use the TO *<mdx filename>* option, the index tags are written to the specified .MDX file. You can create up to 47 individual index tags in a single .MDX file.

The .NDX files you want to copy and the associated table must already be open before you issue the COPY INDEXES command. In a multiuser environment, the table associated with the indexes you want to copy must be opened in exclusive mode.

Example

The following example first creates two .NDX indexes for the Company table. It then uses COPY INDEXES to create tags for the current table using these two existing .NDX files as the source:

```
USE Company EXCLUSIVE
INDEX ON Company TO Company   && create .ndx
INDEX ON Compcode TO Compcode && create .ndx
* There are now two .NDX indexes
SET INDEX TO Company, Compcode
COPY INDEXES Company, Compcode
SET ORDER TO TAG Company
BROWSE FIELDS Company, Compcode
```

Portability

Not supported in dBASE III PLUS.

See Also

COPY TAG, DELETE TAG, INDEX, MDX(), NDX(), SET INDEX, SET ORDER, TAG(), TAGNO(), TAGCOUNT(), USE

COPY MEMO

Fields and records

Copies the contents of the specified memo field to a file.

Syntax

COPY MEMO *<memo field>* TO *<filename>* | ?
[ADDITIVE]

<memo field> Specifies the memo field to copy.

TO *<filename>* | ? Specifies the name of the text file where text will be copied. By default, Visual dBASE assigns a .TXT extension and saves the file in the current directory. The ? option displays a dialog box, in which you specify the name of the target file and the directory to save it in.

ADDITIVE Appends the contents of the memo field to the end of an existing text file. Without the ADDITIVE option, *Visual* dBASE overwrites any previous text in the text file.

C

Description

Use COPY MEMO to export memo file text in the current record to a text file. You can also use COPY MEMO to copy images or other binary-type data to a file; however, binary fields are recommended for storing images, sound, and other user-defined binary information.

If you specify the ADDITIVE option, *Visual* dBASE appends the contents of the memo field to the end of the named file, which lets you combine the contents of memo fields from more than one record. When you don't use ADDITIVE, dBASE displays a warning message before overwriting an existing file if SET SAFETY is ON. You can store only a single image in either a memo field or in a file, so do not use the ADDITIVE option of COPY MEMO when copying an image to a file. (RESTORE IMAGE can display an image stored in either a memo field or a text file.)

Example

The following example uses COPY MEMO to create a text file that holds the unformatted contents of all Notes memo fields in the Company table:

```
USE Company
IF .NOT. FILE("Test.Txt")
    COPY MEMO Notes TO Test.Txt
    SKIP
ENDIF
DO WHILE .NOT. EOF()
    COPY MEMO Notes TO Test.TXT ADDITIVE
    SKIP
ENDDO
MODI COMM Test.TXT      && Note all memos in .TXT file
RETURN
```

Portability

Not supported in dBASE III PLUS.

See Also

APPEND MEMO, COPY, COPY BINARY, COPY FILE, REPLACE BINARY, REPLACE OLE

COPY STRUCTURE

Table basics

Creates an empty table with the same structure as the current table.

Syntax

```
COPY STRUCTURE TO <filename> | ? | <filename skeleton>
[[TYPE] PARADOX | DBASE]
[FIELDS <field list>]
[[WITH] PRODUCTION]
```

<filename> | ? | <filename skeleton> The name of the table you want to create. COPY STRUCTURE TO ? and COPY STRUCTURE TO <filename skeleton> display a dialog box, in which you can specify the name of the destination table. If you specify a table name without including its path, *Visual* dBASE saves the table to the current drive and directory. If you specify a table name without including an extension, defining a default table type with SET DBTYPE, or using one of the TYPE options, *Visual* dBASE assigns a .DBF extension.

You can also copy a structure to a table in a database (defined using the BDE Configuration Utility) by specifying the database as a prefix (enclosed in colons) to the name of the table, that is, *:database name:table name*. If the database is not already open, *Visual* dBASE displays a dialog box in which you specify the parameters, such as a login name and password, necessary to establish a connection to that database.

[TYPE] PARADOX | DBASE Specifies the type of table to create, overriding the current setting of DBTYPE. The TYPE keyword is included for readability only; it has no affect on the operation of the command.

Specifying PARADOX creates a Paradox table. with a .DB extension.

Specifying DBASE creates a dBASE table (the default). If you don't include an extension for <filename>, *Visual* dBASE assigns a .DBF extension.

FIELDS <field list> Determines which fields *Visual* dBASE includes in the structure of the new table. The fields appear in the order specified by <field list>.

[WITH] PRODUCTION Creates a production .MDX file for the new table. The new index file has the same index tags as the production index file associated with the original table.

Description

The COPY STRUCTURE command copies the structure of the current table but does not copy any records. If SET SAFETY is OFF, *Visual* dBASE overwrites any existing tables of the same name without issuing a warning message.

The COPY STRUCTURE command copies the entire table structure unless limited by the FIELDS option or the SET FIELDS command. When you issue COPY STRUCTURE without the FIELDS <field list> option, *Visual* dBASE copies the fields in the SET FIELDS TO list to the new table. The _dbaselock field created with the CONVERT command is not copied to new tables.

You can use COPY STRUCTURE to create an empty table structure with fields from more than one table. To do so,

- 1 Open the source tables in different work areas.
- 2 Use the `FIELDS <field list>` option, including the table alias for each field name not in the current table.

You can also set a relationship between the tables using the `SET RELATION` command, and then use `COPY STRUCTURE` to copy fields from the related tables.

Example

The following example uses `COPY STRUCTURE` to copy selected fields from two open tables to a new table:

```
USE Contact
USE Company IN SELECT()
COPY STRUCTURE TO CntctLst ;
  FIELDS Company->Company, ;
  Contact->CompCode, Contact->Contact, ;
  Company->Street1, Company->Street2, ;
  Company->City, Company->State_Prov, ;
  Company->Zip_P_Code
USE CntctLst
APPEND
CLOSE ALL
```

See Also

`APPEND`, `APPEND FROM`, `COPY`, `COPY TO...STRUCTURE EXTENDED`, `DISPLAY STRUCTURE`, `MODIFY STRUCTURE`, `SET FIELDS`, `SET SAFETY`

COPY TABLE

Table basics

Copies a specified table.

Syntax

```
COPY TABLE <source tablename> TO <target tablename>
[[TYPE] DBASE | PARADOX]
```

<source table name> The name of the table that you want to copy. You can also copy a table in a database (defined using the BDE Configuration Utility) by specifying the database as a prefix (enclosed in colons) to the name of the table, that is, *:database name:table name*. If the database is not already open, *Visual dBASE* displays a dialog box in which you specify the parameters, such as a login name and password, necessary to establish a connection to that database.

<target table name> The name of the table you want to create. The table type is the same as the source table. If you copy a table in a database, you must specify the same database as the destination of the target table.

[TYPE] PARADOX | DBASE Specifies the type of table to copy, which can include either Paradox or dBASE tables. Overrides the current setting of `DBTYPE`.

Description

Use the COPY TABLE command to copy tables and their associated .NDX and .MDX index files. For Paradox tables, COPY TABLE copies associated indexes, BLOB and .VAL files. Make sure the table is not in use before you attempt to copy it.

Example

The following example uses COPY TABLE to create a duplicate table without first opening Clients.DBF:

```
COPY TABLE Clients TO Region1
```

Portability

Not supported in dBASE IV or dBASE III PLUS.

See Also

COPY, COPY FILE, DELETE FILE, DELETE TABLE, ERASE

COPY TAG

Table organization

Copies .MDX index tags to .NDX files. (This command is only applicable for dBASE tables.)

Syntax

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

<tag name> Specifies a .MDX index tag you want to copy.

OF <.mdx filename> The .MDX file that contains the specified index tag. If you specify a file without including its path, *Visual* dBASE looks for the file in the current directory, then in the path you specify with SET PATH. If you specify a file without including its extension, *Visual* dBASE assumes an .MDX extension.

TO <.ndx filename> The name of an .NDX file that you want to copy the index tag to. If you specify a file without including its path, *Visual* dBASE creates the file in the current directory. If you specify a file without including its extension, *Visual* dBASE assigns an .NDX extension.

Description

COPY TAG creates individual .NDX files from the index tags in the production index or a specified .MDX file. You might use this command if tables and indexes you create might be accessed by previous versions of dBASE.

Before issuing the COPY TAG command, the .MDX file and its associated table must already be open. Only one tag can be copied at a time. A FOR clause of an .MDX file tag is ignored, since .NDX files do not support them.

Example

The following example uses COPY TAG to create an .NDX index file from an open tag index:

```
DELETE FILE Company.ndx && remove any existing index
USE Company EXCLUSIVE
INDEX ON Company TAG Company
* now there is a TAG index
COPY TAG Company TO Company
* COPY TAG creates a .NDX index
SET INDEX TO Company
BROWSE FIELDS Company, Compcode
```

C

Portability

Not supported in dBASE III PLUS.

See Also

COPY INDEXES, FOR(), INDEX, MDX(), NDX(), SET INDEX, SET ORDER, TAG(), TAGCOUNT(), TAGNO()

COPY TO ARRAY

Fields and records

Copies data from non-memo fields of the current table, overwrites elements of an existing array, and moves the record pointer to the last record copied.

Syntax

```
COPY TO ARRAY <array name>
[<scope>]
[FOR <condition 1>]
[WHILE <condition 2>]
[FIELDS <field list>]
```

<scope> The number of records to copy to the specified array. RECORD <*n*> identifies a single record by its record number. (You can also specify a bookmark for tables that do not provide record numbers.) NEXT <*n*> identifies *n* records, beginning with the current record. ALL specifies all records. REST specifies all records from the current record to the end of the file.

FOR <condition 1>

WHILE <condition 2> Determines which records are affected by COPY TO ARRAY. FOR restricts COPY TO ARRAY to records that meet <condition 1>. WHILE starts processing with the current record and continues with each subsequent record as long as <condition 2> is true.

FIELDS <field list> Copies data from the fields in <field list> in the order of <field list>. Without FIELDS, dBASE copies all the fields the array can hold in the order they occur in the current table.

Description

Use COPY TO ARRAY to copy records from the current table to an existing array. To copy specified records and fields, use DECLARE Sample[<memvar 1>,<memvar 2>], where <memvar 1> is the maximum number of records to copy and <memvar 2> is the maximum number of fields to copy. If you use DECLARE Sample[2,3], you can copy a maximum of three fields from a maximum of two records. To copy data from all fields in all records in a table without memo fields, use DECLARE sample[RECCOUNT(),FCOUNT()].

Using a two-dimensional array, the first subscript is the number of records to copy, and the second subscript is the number of fields to copy. For example, if you DECLARE Test[2,3], COPY TO ARRAY Test can copy three fields from two records.

COPY TO ARRAY can copy to arrays that have more than two dimensions (multi-dimensional), but it uses only the last two subscripts. The next-to-last subscript in <array name> determines how many records the array can hold, and the last subscript determines how many fields from each record the array can hold. For example, DECLARE Test[4,5,2,3] and DECLARE Test[2,3] both create an array to which COPY TO ARRAY copies three fields from two records.

If the array is one-dimensional, COPY TO ARRAY can copy only one record; in this case, the number of elements determines the number of fields to copy. For example, if you use DECLARE Test[5], COPY TO ARRAY copies the first five fields from the current record.

COPY TO ARRAY copies in record number or index order and, within each record, in field number order unless you use the FIELDS option to specify the order of the fields to copy.

If the number specified by the last subscript in the array is larger than the number of fields copied from the table, the additional array elements remain initialized to .F. or to a previously stored value. Similarly, if the number specified by the next-to-last subscript of the array is larger than the number of records copied from the table, the additional array elements remain initialized to .F. or to a previously stored value.

Example

The following example uses COPY TO ARRAY to copy 5 records of a view created with SET RELATION to an array of 5 rows and 3 columns for further manipulation:

```
CLEAR
SET TALK OFF
USE Contact ORDER Compcode IN SELECT( )
USE Company IN SELECT( )
SELECT Company
SET RELATION TO CompCode INTO Contact    && Set a relation on CompCode
get_rec = 5
DECLARE CompCtc[get_rec,3]              && Create an array 5 records long with 3 fields
COPY TO ARRAY CompCtc ;
    NEXT get_rec FIELDS Company->Company, ;
    Contact->Contact, Company->YTD_Sales
Record = 1
? "Company" AT 10, "Contact" AT 40, ;
  "Sales" AT 60
```



```

?
DO WHILE Record <= get_rec
  ? CompCtc[Record,1] AT 10, CompCtc[Record,2] AT 40,;
  LTRIM(STR(CompCtc[Record,3])) AT 60
  Record = Record + 1
ENDDO
CLOSE ALL

```

C

See Also

APPEND FROM ARRAY, DECLARE, REPLACE FROM ARRAY, SET FIELDS, STORE MEMO

COPY TO...STRUCTURE EXTENDED

Table basics

Creates a new table whose records contain the structure of the current table.

Syntax

```

COPY TO <filename> | ?
STRUCTURE EXTENDED
[[TYPE] PARADOX | DBASE]

```

or

```

COPY STRUCTURE EXTENDED TO <filename> | ?
[[TYPE] PARADOX | DBASE]

```

<filename> | ? The name of the table that you want to create to contain the structure of the current table. COPY TO ? and COPY STRUCTURE EXTENDED TO ? display a dialog box in which you can specify the name of the destination table. If you specify a table name without including its path, *Visual dBASE* saves the table to the current drive and directory. If you specify a table name without including an extension, defining a default table type with SET DBTYPE, or using one of the TYPE options, *Visual dBASE* assigns a .DBF extension.

You can also create a table in a database (defined using the BDE Configuration Utility) by specifying the database as a prefix (enclosed in colons) to the name of the table, that is, *:database name:table name*. If the database is not already open, *Visual dBASE* displays a dialog box in which you specify the parameters, such as a login name and password, necessary to establish a connection to that database.

[[TYPE] PARADOX | DBASE Specifies the type of table to create. The TYPE keyword is included for readability only; it has no effect on the operation of the command.

Specifying PARADOX creates a Paradox table with a .DB file extension.

Specifying DBASE creates a dBASE table (the default). If you don't include an extension for <filename>, *Visual dBASE* assigns a .DBF extension.

Description

COPY TO...STRUCTURE EXTENDED copies the structure of the current table to records in a new table.

COPY TO...STRUCTURE EXTENDED first defines a table, called a structure-extended table, containing five fields of fixed names, types, and lengths. Once the structure-extended table is defined, COPY TO...STRUCTURE EXTENDED appends records that provide information about each field in the current table. The fields in the structure-extended table store the following information about fields in the current table:

Field	Contents
FIELD_NAME	Character field that contains the name of the field.
FIELD_TYPE	Character field that contains the field's data type.
FIELD_LEN	Numeric field that contains the field length.
FIELD_DEC	Numeric field that contains the number of decimal places for numeric and float data.
FIELD_IDX	Character field (not created in dBASE III PLUS) that indicates if index tags were created on particular fields when the current table was created.

When the process is complete, the structure-extended table contains as many records as there are fields in the current table. You can then use CREATE...FROM to create a new table from the information provided by the structure-extended table.

The _dbaselock field created with the CONVERT command is not copied to structure-extended tables.

Example

The following example uses COPY TO ... STRUCTURE EXTENDED to create the table Names that contains the structure of Clients as its records. The structure is then altered by deleting those records beyond the Zip field. CREATE is then used to create a new table with the abbreviated structure, and records from Clients are appended. Creating a new table via COPY TO ... STRUCTURE EXTENDED is also a useful tool for correcting .DBF file corruption:

```
SET TALK OFF
SET SAFETY OFF
USE Clients
COPY TO Names STRUCTURE EXTENDED
USE Names EXCLUSIVE
DELETE FOR RECNO(>)>7           && Excludes fields beyond Zip
PACK
CREATE Names2 FROM Names
DISPLAY STRUCTURE               && Names2 currently in use
? ALIAS()+" has " + LTRIM(STR(RECCOUNT())) + " records at this point."
WAIT
APPEND FROM Clients
?
? ALIAS()+" has " + LTRIM(STR(RECCOUNT())) + " records now."
WAIT
GO TOP
BROWSE
```

See Also

COPY, COPY STRUCTURE, CREATE, CREATE...FROM, CREATE...STRUCTURE EXTENDED, DISPLAY STRUCTURE, LIST STRUCTURE, MODIFY STRUCTURE, SET SAFETY

COS()

Numeric data

C

Returns the trigonometric cosine of an angle.

Syntax

`COS(<expN>)`

<expN> The size of the angle in radians. To convert an angle's degree value to radians, use `DTOR()`. For example, to find the cosine of a 30-degree angle, use `COS(DTOR(30))`.

Description

`COS()` calculates the ratio between the side adjacent to an angle and the hypotenuse in a right triangle. `COS()` returns a float from -1 to $+1$. `COS()` returns 0 when `<expN>` is $\pi/2$ or $3\pi/2$ radians.

Use `SET DECIMALS` to set the number of decimal places `COS()` displays.

The secant of an angle is the reciprocal of the cosine of the angle. To return the secant of an angle, use `1/COS()`.

Example

The following example uses `COS()` to compute the length of a rafter after room width and roof angle have been entered by the user:

```
SET PROCEDURE TO PROGRAM(1) ADDITIVE
LOCAL f
f = NEW PFORM()
f.Open()

CLASS PFORM OF FORM
    this.HelpFile = ""
    this.Width = 43.00
    this.Height = 10.47
    this.Left = 46.60
    this.Text = "Rafter Computer"
    this.Top = 6.94
    this.HelpId = ""

    DEFINE ENTRYFIELD ROOM OF THIS;
        PROPERTY;
            Width 4.00;;
            Picture "999",;
            Height 1.00;;
            Left 32.00;;
            Top 2.00;;
            Border .T.,;
            Value 0

    DEFINE ENTRYFIELD ANGLE OF THIS;
        PROPERTY;
            Width 4.00;;
            Picture "99",;
            Height 1.00;;
```

COS()

```
Left      32.00;;
Top       4.00;;
Border .T.;;
Value     0

DEFINE ENTRYFIELD RAFTLENGTH OF THIS;
PROPERTY;
Width     6.00;;
Height    1.00;;
Left      33.00;;
Top       6.00;;
Border .T.;;
Value     0.00;;
OnGotFocus RESULTS

DEFINE TEXT LN1 OF THIS;
PROPERTY;
Width     26.00;;
Height    1.00;;
Left      3.00;;
Text "Enter Width of Room:",";
ColorNormal "N/W",;
Border .F.

DEFINE TEXT LN2 OF THIS;
PROPERTY;
Width     27.00;;
Height    1.00;;
Left      3.00;;
Text "Enter Angle of Rise:",";
Top       4.00;;
ColorNormal "N/W",;
Border .F.

DEFINE TEXT LN3 OF THIS;
PROPERTY;
Width     29.00;;
Height    1.00;;
Left      3.00;;
Text "Cut Rafter to this length:",";
Top       6.00;;
ColorNormal "R/W",;
Border .F.

DEFINE PUSHBUTTON EXIT OF THIS;
PROPERTY;
Width     15.60;;
Default .T.;;
OnClick {;Form.Close();};
Height    1.12;;
Left      14.00;;
Text "Exit",;
Top       8.00;;
ColorNormal "N/W"
```

```

ENDCLASS

FUNCTION Results
Form.RaftLength.Value=LTRIM(STR((Form.Room.Value/2);
/COS(DTOR(Form.Angle.Value)),6,2))
RETURN .T.

```

C

Portability

Not supported in dBASE III PLUS.

See Also

ACOS(), DTOR(), PI(), RTOD(), SET DECIMALS, SIN(), TAN()

COUNT

Fields and records

Counts the number of table records that match specified conditions.

Syntax

```

COUNT
[<scope>]
[FOR <condition 1>]
[WHILE <condition 2>]
[TO <memvar>]

```

<scope> The number of records to count. RECORD <*n*> identifies a single record by its record number. NEXT <*n*> identifies *n* records, beginning with the current record. ALL specifies all records. REST specifies all records from the current record to the end of the file.

FOR <condition 1>

WHILE <condition 2> Determines which records are affected by COUNT. FOR restricts COUNT to records that meet <condition 1>, starting at the first record of the table or scope and continuing to the end of the table or scope. WHILE starts processing with the current record and continues with each subsequent record as long as <condition 2> is true.

TO <memvar> Stores the result of COUNT, a number, to the specified memory variable.

Description

Use COUNT to total the number of records that meet a specified criterion. If SET TALK is ON, the COUNT command also displays the total. If SET DELETED is ON, records marked for deletion are not included in the count.

In a multi-user environment, COUNT automatically locks the table during its operation if SET LOCK is ON (the default), and unlocks it after the count is finished. If SET LOCK is OFF, you can still perform a count; however the result may change if another user changes the table.

You can also count the total number of records in a table using the RECCOUNT() function. However, unlike COUNT, RECCOUNT() does not let you specify conditions to qualify the records it counts.

Example

The following example uses COUNT to determine the number of records for companies from California with a value entered for YTD_Sales in the Company table. This number is then used to calculate an average sales figure for all California companies with a current YTD_Sales entry:

```
SET TALK OFF
CLEAR
USE Company
Condition = "State_Prov = 'CA'"
CALCULATE SUM(YTD_Sales) TO Sales FOR &Condition
COUNT TO Num FOR &condition .AND. YTD_SALES<>0
? "Average California Sales: $" + LTRIM(STR(Sales/Num))
CLOSE ALL
```

See Also

AVERAGE, CALCULATE, RECCOUNT(), SUM, TOTAL

CREATE

Table basics

Opens the Table Designer to create or modify a table interactively.

Syntax

```
CREATE
[<filename> | ? | <filename skeleton>]
[[TYPE] PARADOX | DBASE]
[EXPERT [PROMPT]]
```

<filename> | ? | <filename skeleton> The name of the table you want to create. CREATE ? and CREATE <filename skeleton> display a dialog box, in which you can specify the name of a new table. If you specify a table name without including its path, *Visual* dBASE saves the table to the current drive and directory. If you specify a table name without including an extension, *Visual* dBASE assigns a .DBF extension or the file extension of the table type you specified with SET DBTYPE. If you don't specify a name, the table remains untitled until you save the file. If you specify an existing table name, *Visual* dBASE asks whether you want to modify the existing table or overwrite it.

You can also create a table in a database (defined using the BDE Configuration Utility) by specifying the database as a prefix (enclosed in colons) to the name of the table, that is, *:database name:table name*. If the database is not already open, *Visual* dBASE displays a dialog box in which you specify the parameters, such as a login name and password, necessary to establish a connection to that database.

[TYPE] PARADOX | DBASE Specifies the type of table to create. The TYPE keyword is included for readability only; it has no effect on the operation of the command.

Specifying PARADOX creates a Paradox table with a .DB extension.

Specifying DBASE creates a dBASE table (the default). If you don't include an extension for *<filename>*, dBASE assigns a .DBF extension.

C

Description

CREATE opens the Table Designer, an interactive environment in which you can create or modify the structure of a table, or the Table Expert, a tool which guides you through the process of creating tables. The type of table you create or modify depends on your selection of the table type specified with the CREATE command, or with SET DBTYPE.

Create a table by defining the name, type, and size of each field. For more information on using the Table Designer, see the *User's Guide*.

Example

The following examples show several ways to use CREATE to design a table from the Command window:

```
CREATE MailList           && Opens table designer -.DBF table
CREATE MailList TYPE PARADOX
                           && Opens table designer - .DB table
CREATE ?                  && Opens dialog box for naming file
```

See Also

APPEND, APPEND MEMO, COPY STRUCTURE, DISPLAY STRUCTURE, LIST STRUCTURE, MODIFY STRUCTURE, REPLACE

CREATE APPLICATION

Forms

Opens the Form Designer to create or modify a form interactively.

Syntax

```
CREATE APPLICATION
[<filename> | ? | <filename skeleton>]
[EXPERT [PROMPT]]
```

<filename> | ? | <filename skeleton> The form to create or modify. CREATE APPLICATION ? and CREATE APPLICATION *<filename skeleton>* display a dialog box, from which you can select a file. If you specify a file without including its path, dBASE looks for the file in the current directory, then in the path you specify with SET PATH. If you specify a file without including its extension, dBASE assumes .WFM.

[EXPERT [PROMPT]] If the PROMPT clause is used, a dialog appears asking if you want to use the Form Designer or the Form Expert. You can then invoke either the designer or the expert. The EXPERT clause without PROMPT causes the Form Expert to be invoked.

Description

CREATE SCREEN, CREATE APPLICATION, and CREATE FORM are identical; all launch the Form Designer. See the description of CREATE FORM for more information.

Example

See the example of CREATE FORM. In this example, CREATE APPLICATION works the same as CREATE FORM.

Portability

Not supported in dBASE III PLUS. In dBASE IV, CREATE APPLICATION launches the dBASE IV Applications Generator.

See Also

CREATE FORM, CREATE SCREEN, MODIFY APPLICATION, MODIFY FORM, MODIFY SCREEN, OPEN FORM

CREATE CATALOG

Table basics

Creates a catalog file.

Syntax

CREATE CATALOG

[<filename> | ? | <filename skeleton>

<filename> | ? | <filename skeleton> The catalog file you want to create. CREATE CATALOG ? and CREATE CATALOG <filename skeleton> display a dialog box, in which you can specify the name of a catalog file.

Description

Use CREATE CATALOG to create a new catalog. Catalog file names are limited to eight characters, following rules for naming DOS files. *Visual* dBASE always assigns the extension (.CAT) to catalog file names.

If you set TITLE ON when you create a new catalog, *Visual* dBASE prompts you to enter a one-line description of the catalog file. A master catalog, CATALOG.CAT, stores catalog file names along with catalog title descriptions. The description you enter for each catalog entry is displayed later in the catalog window when you use the SET CATALOG TO ? command to select a catalog name.

Catalogs are dBASE tables which have a predefined table structure. When you create or select a catalog file, it is automatically opened in its own work area buffer and SET CATALOG is set ON. From then on, all tables and associated files such as index, query, format, report, and label files you use or new files you create are added to the catalog.

Catalog structure

All catalogs have the same structure. The following table describes the fields contained in a catalog.

Field	Field name	Type	Width	Description
1	Path	Character	70	Contains the full path name of the table or file if it is not in the current directory.
2	File_name	Character	12	This is the name of the file, including its extension.
3	Alias	Character	8	Assigned table alias. If you do not assign an alias name, <i>Visual</i> dBASE uses the table name as the alias name. This field is left blank for all other file types.
4	Type	Character	3	This field contains the default file-name 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.
5	Title	Character	80	This is an optional field. If SET TITLE is ON, <i>Visual</i> dBASE prompts you to add a description of up to 80 characters when you create the catalog.
6	Code	Numeric	3	When you have a catalog open, Code is the number assigned to each table in use. Program files are assigned a value of 0. A table is assigned a number when it is created. Each new table is assigned the next available number higher than the previous table. Files associated with a table, such as an index, format, label, query, report form, screen, and view files, are assigned the same code number as the table they reference.
7	Tag	Character	4	Field not currently used.

With the exception of the Title and Tag fields, *Visual* dBASE automatically fills in the field contents whenever a new file is entered in the catalog.

If you want to modify the contents of the catalog table, first open it in a user-accessible work area, and then use EDIT or BROWSE. For more information, see SET CATALOG.

Example

The following example uses CREATE CATALOG to make a CATALOG for your files.

```
CREATE CATALOG Mail      && Creates a Catalog called Mail. Prompts you ;
                        to enter a description of this Catalog
CREATE CATALOG ?        && Displays Open Catalog dialog box
```

Portability

Not supported in dBASE IV or dBASE III PLUS.

See Also

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

CREATE COMMAND

Displays a specified program file for editing, or displays an empty editing window.

Syntax

```
CREATE COMMAND
[<filename> | ? | <filename skeleton>]
[WINDOW <window name>]
```

<filename> | ? | <filename skeleton> The file to display and edit. The ? and <filename skeleton> options display a dialog box from which you can select a file. If you specify a file without including its path, dBASE looks for the file in the current directory, then in the path you specify with SET PATH. If you specify a file without including its extension, dBASE assumes .PRG. If you issue CREATE COMMAND without an option, dBASE displays an empty editing window.

[WINDOW <window name>] Included for compatibility with dBASE IV. Displays the file in <window name>, previously opened with ACTIVATE WINDOW.

Default

By default, CREATE COMMAND launches the dBASE internal Text Editor. You can specify an alternate editor by changing the EDITOR setting in DBASEWIN.INI. To do so, either use the SET command to specify the setting interactively, or enter the EDITOR parameter directly in DBASEWIN.INI.

Description

Use CREATE COMMAND to create new or edit existing program files. Use DO to execute program files.

Note dBASE compiles programs before running them, and assigns the compiled files the same name as the original, but with the letter "O" as the last letter. For example, the compiled version of SALESRPT.PRG would be SALESRPT.PRO. If SALESRPT.PRO already exists, it is overwritten. For this reason, avoid using file names ending in "O" in directories containing compiled programs.

If you're creating a new program file, CREATE COMMAND displays an editing window with no file. If you specify an existing file, CREATE COMMAND displays the program in an editing window, with the cursor at the beginning of line 1.

The dBASE menu changes when an editing window is active. See the *User's Guide* for information on the internal Text Editor, including its menu and shortcut keys.

Portability

Not supported in dBASE IV, but MODIFY COMMAND is supported in dBASE IV.

See Also

DO, CREATE FILE, SET DEVELOPMENT, SET EDITOR

CREATE FILE

Disk and file utilities

C

Displays a specified text file for editing, or displays an empty editing window.

Syntax

CREATE FILE

[<filename> | ? | <filename skeleton>]

[WINDOW <window name>]

<filename> | ? | <filename skeleton> The text file to display and edit. The ? and <filename skeleton> options display a dialog box from which you can select a file. If you specify a file without including its path, dBASE looks for the file in the current directory, then in the path you specify with SET PATH. If you specify a file without including its extension, dBASE assumes .PRG. If you issue CREATE FILE without an option, dBASE displays an empty editing window.

[WINDOW <window name>] Included for compatibility with dBASE IV. Displays the file in <window name>, previously opened with ACTIVATE WINDOW.

Description

Use CREATE FILE to create and edit text files using the *Visual* dBASE internal text editor, or the editor specified in SET EDITOR.

CREATE FILE is identical to CREATE COMMAND, MODIFY COMMAND and MODIFY FILE. See CREATE COMMAND for more information.

Portability

Not supported in dBASE IV, but MODIFY FILE is supported in dBASE IV.

See Also

CREATE COMMAND, SET EDITOR

CREATE FORM

Forms

Opens the Form Designer to create or modify a form.

Syntax

CREATE FORM

[<filename> | ? | <filename skeleton>]

[EXPERT [PROMPT]]

<filename> | ? | <filename skeleton> The form to create or modify. CREATE FORM ? and CREATE FORM <filename skeleton> display a dialog box, from which you can select a file. If you specify a file without including its path, dBASE looks for the file in the current directory, then in the path you specify with SET PATH. If you specify a file without including its extension, dBASE assumes .WFM.

[EXPERT [PROMPT]] If the PROMPT clause is used, a dialog appears asking if you want to use the Form Designer or the Form Expert. You can then invoke either the designer or the expert. The EXPERT clause without PROMPT causes the Form Expert to be invoked.

Description

Use CREATE FORM to open the Form Designer or Form Expert and create or modify a form interactively. The Form Designer automatically generates dBASE program code that defines the contents and format of a form, and stores this code in an editable text file (.WFM).

CREATE SCREEN, CREATE APPLICATION and CREATE FORM are identical. For all these commands, the presence of a form file determines whether a create or modify operation occurs. If the .WFM file exists, the commands let you modify it in the Form Designer. If the file doesn't exist, the commands create a new file.

Since a .WFM file is a program file, you can edit it with MODIFY COMMAND.

See the Forms chapters in the *User's Guide* for instructions on using the Form Designer.

Note The Forms Designer is a two-way tool. You can open a form in the Form Designer even if you've edited the code in the .WFM file.

Example

The following examples open the Save File dialog box with the cursor positioned at the File Name block to name a new form. The picklist of .WFM files on the current directory is available if you desire to use an existing form name to create a new form. By contrast, MODIFY FORM ? would edit an existing form:

```
CREATE FORM ?
CREATE FORM *.WFM
```

CREATE FORM issued alone in the Command window will open an unnamed form design surface:

```
CREATE FORM
```

Portability

Not supported in dBASE IV or dBASE III PLUS.

See Also

CREATE APPLICATION, CREATE SCREEN, MODIFY APPLICATION, MODIFY FORM, MODIFY SCREEN, OPEN FORM

CREATE LABEL

Input/Output

C

Opens the Report Designer to create or modify a label file.

Syntax

CREATE LABEL

[<filename> | ? | <filename skeleton>]

[EXPERT [PROMPT]]

<filename> | ? | <filename skeleton> The label file to create or modify. The ? and <filename skeleton> options display a dialog box from which you can select a file. If you specify a file without including its path, dBASE looks for the file in the current directory, then in the path you specify with SET PATH. If you specify a file without including its extension, dBASE assumes .RPL.

If dBASE can't find <filename>, it creates the file. By default, dBASE assigns an .RPL extension to <filename> and saves the file in the current directory.

CREATE LABEL without an option opens the empty Report Designer to create a new label file.

[EXPERT [PROMPT]] If the PROMPT clause is used, a dialog appears asking if you want to use the Report Designer or the Report Expert. You can then invoke either the designer or the expert. The EXPERT clause without PROMPT causes the Report Expert to be invoked.

Description

Use CREATE LABEL to open the Report Designer and create or modify a label file. A label file contains the information that formats labels. For information about using the Report Designer, see the Crystal Reports documentation. CREATE LABEL and MODIFY LABEL are identical commands.

Before issuing CREATE LABEL, you must have a default printer selected. After you create or modify the label file, use LABEL FORM to print the labels.

Example

This example opens a database table and then issues CREATE LABEL to construct a label form:

```
CLOSE DATABASE
USE Company
CREATE LABEL COMPLBL1
* If COMPLBL1.LBL already exists then it will be modified.
* Otherwise, a new label form will be created.
```

Portability

The <filename skeleton> option is not supported in dBASE IV or dBASE III PLUS. In dBASE IV and dBASE III PLUS, the default label file extension is .LBL.

See Also

CREATE REPORT, LABEL FORM

CREATE MENU

Forms

Opens the Menu Designer to create or modify a menu file.

Syntax

CREATE MENU

[<filename> | ? | <filename skeleton>]

<filename> | ? | <filename skeleton> The menu file to create or modify. CREATE MENU? and CREATE MENU <filename skeleton> display a dialog box, from which you can select a file. If you specify a file without including its path, dBASE looks for the file in the current directory, then in the path you specify with SET PATH. If you specify a file without including its extension, dBASE assumes .MNU.

Description

Use CREATE MENU to design a menu for a form.

The menu you design is stored in a menu definition file (.MNU), which contains dBASE program code. You attach this program to a form via the form's Menu property.

For information on using the Menu Designer, see the *User's Guide*.

See Also

CLASS MENU

CREATE POPUP

Forms

Opens the Menu Designer to create or modify a popup menu file.

Syntax

CREATE POPUP

[<filename> | ? | <filename skeleton>]

<filename> | ? | <filename skeleton> The popup menu file to create or modify. CREATE POPUP ? and CREATE POPUP <filename skeleton> display a dialog box, from which you can select a file. If you specify a file without including its path, dBASE looks for the file in the current directory, then in the path you specify with SET PATH. If you specify a file without including its extension, dBASE assumes .POP.

Description

Use CREATE POPUP to design a popup menu for a form. The menu you design is stored in a popup definition file (.POP), which contains dBASE program code. For information on using the Menu Designer, see UG_MENU.

See Also

CLASS MENU, CLASS POPUP

CREATE QUERY

Table organization

C

Opens the Query Designer to create or modify a query file.

Syntax

CREATE QUERY

[<filename> | ? | <filename skeleton>]

<filename> | ? | <filename skeleton> Specifies the name of the file you want to create or modify. CREATE QUERY ? and CREATE QUERY <filename skeleton> display a dialog box, in which you can specify the name of a query file. If you specify a file without including its path, *Visual* dBASE saves the file to the current drive and directory. If you specify a file without including an extension, *Visual* dBASE assumes a .QBE extension.

Description

CREATE QUERY performs the same operation as CREATE VIEW. Use either command to invoke the Query Designer, an interactive environment in which you can create or modify a query. The Query Designer creates a .QBE file that displays only the fields and records that meet specified conditions. To activate a query (.QBE) file, you can use the SET VIEW command. For more information on using the Query Designer, see the *User's Guide*.

Example

CREATE QUERY or CREATE VIEW with no arguments opens the Query Designer and generates an untitled query, which the user will name later:

```
CREATE QUERY
```

CREATE QUERY followed by a file name creates a file of that name when the file is saved:

```
CREATE QUERY Myquery
```

If MYQUERY.QBE already exists, you are asked if you want to overwrite it.

See Also

MODIFY, SET VIEW

CREATE REPORT

Input/Output

Opens the Report Designer to create or modify a report file.

Syntax

CREATE REPORT

[CROSSTAB]

[<filename> | ? | <filename skeleton>]

[EXPERT [PROMPT]]

CROSSTAB Opens the Report Designer with the Cross-Tab dialog box displayed.

<filename> | ? | <filename skeleton> The report file to create or modify. The ? and <filename skeleton> options display a dialog box from which you can select a file. If you specify a file without including its path, dBASE looks for the file in the current directory, then in the path you specify with SET PATH.

If you specify a file without including its extension, dBASE assumes .RPT if you haven't specified CROSSTAB, and .RPC if you have. If dBASE can't find <filename>, it creates a file with the appropriate extension and saves it in the current directory.

CREATE REPORT without any options opens the empty Report Designer to create a new report.

[EXPERT [PROMPT]] If the PROMPT clause is used, a dialog appears asking if you want to use the Report Designer or the Report Expert. You can then invoke either the designer or the expert. The EXPERT clause without PROMPT causes the Report Expert to be invoked.

Description

Use CREATE REPORT to open the Report Designer and create or modify a report file. A report file contains the information that formats a report. For information about using the Report Designer, see the Crystal Reports documentation. CREATE REPORT and MODIFY REPORT are identical commands.

Before issuing CREATE REPORT, you must have a default printer selected. After you create or modify the report file, use REPORT FORM to print the report.

Example

This example opens a database table and then issues CREATE REPORT to construct a report form:

```
CLOSE DATABASE
USE COMPANY
CREATE REPORT Compre1
* If Compre1 already exists then it will be modified.
* Otherwise, a new report form will be created.
```

Portability

The <filename skeleton> option is not supported in dBASE IV or dBASE III PLUS. In dBASE IV and dBASE III PLUS, the default report file extension is .FRM.

See Also

REPORT FORM

CREATE SCREEN

Forms

C

Opens the Form Designer to create or modify a form.

Syntax

CREATE SCREEN

[<filename> | ? | <filename skeleton>]

[EXPERT [PROMPT]]

<filename> | ? | <filename skeleton> The form to create or modify. CREATE SCREEN ? and CREATE SCREEN <filename skeleton> display the Save File dialog box, from which you can select a file. If you specify a file without including its path, dBASE looks for the file in the current directory, then in the path you specify with SET PATH. If you specify a file without including its extension, dBASE assumes .WFM.

[EXPERT [PROMPT]] If the PROMPT clause is used, a dialog appears asking if you want to use the Form Designer or the Form Expert. You can then invoke either the designer or the expert. The EXPERT clause without PROMPT causes the Form Expert to be invoked.

Description

CREATE SCREEN, CREATE APPLICATION, and CREATE FORM are identical; all launch the Form Designer. See the description of CREATE FORM for more information.

Example

See the example of CREATE FORM. In this example, CREATE SCREEN works the same as CREATE FORM.

Portability

In dBASE III PLUS and dBASE IV, CREATE SCREEN opens the dBASE III PLUS or dBASE IV Form Designer. These Form Designers create two files, .SCR and .FMT. You can use .FMT files in *Visual* dBASE; however, you can't modify them with the *Visual* dBASE Form Designer.

See Also

CREATE APPLICATION, CREATE FORM, MODIFY APPLICATION, MODIFY FORM, MODIFY SCREEN, OPEN FORM

CREATE SESSION

Environment

Creates a new *session* and immediately selects it. Subsequent commands that are session-based apply to the new session.

Syntax

CREATE SESSION

Description

Use CREATE SESSION to initiate a clean work session within dBASE. A session can be compared to a multi-user environment; each session manages its own set of work areas. Associating a form or application with a session ensures that no other form or application can interfere with the environment in the session. For example, if an application running in another session issues CLOSE ALL, the files in the current session are not affected.

In particular, CREATE SESSION does the following:

- Makes all work areas available, even if other running programs have tables open. As in a multi-user environment, the same table can be opened in different sessions, but the record pointer actions are independent in each session. When updates to a table are performed in one session, however, the changes are reflected in the same table opened in another session.
- Resets the value of most SET commands to their default value (a complete list is included later in this section)

Note CREATE SESSION does not release or remove access to memory variables. Access to variables is controlled only by their scope (public, private, local, or static), not by the session in which they are declared.

You might want to issue CREATE SESSION at the following times in an application:

- At the beginning of any program or procedure that needs to start in the default environment, or that needs to be protected from commands issued in another session.
- At the beginning of the first program in a sequence of related programs; this ensures that your program starts in a known environment, and is protected from actions in other sessions.
- At the beginning of the code (.PRG or .WFM) defining a form that is linked to a table; this ensures that the form has access to the table, even if the table is in use in another session.
- Before opening a table you want to open and display in a BROWSE or EDIT window; this makes it possible for you to set filters, indexes, and other conditions relevant to the current view, even if the table is in use in another session.
- Before beginning a series of transactions with BEGINTRANS(); this ensures the transaction can be started, even if another BEGINTRANS() is active in another session.

You can't issue a command to select or end a particular session. A session is selected in one of the following ways:

- When you issue CREATE SESSION, that session is selected.
- When a form is activated, the session in effect when the form was created is selected.
- When you open a Browse or Query window, the session in effect when the Browse or Query was created is selected.

A session ends automatically when all windows or forms associated with the session cease to exist.

The following SET commands affect only the current session:

C

SET AUTOSAVE	SET BLOCKSIZE	SET CARRY
SET CENTURY	SET CONFIRM	SET CUAENTER
SET CURRENCY	SET DATABASE	SET DATE
SET DBTYPE	SET DECIMALS	SET DEFAULT
SET DELETED	SET DELIMITERS	SET DIRECTORY
SET EXACT	SET EXCLUSIVE	SET FIELDS
SET FILTER	SET IBLOCK	SET INDEX
SET KEY TO	SET LOCK	SET MARK
SET MBLOCK	SET MEMOWIDTH	SET NEAR
SET ORDER	SET PATH	SET POINT
SET PRECISION	SET REFRESH	SET RELATION
SET REPROCESS	SET SAFETY	SET SEPARATOR
SET SKIP	SET TALK	SET UNIQUE

For more information on sessions, see Chapter 21 in the *Programmer's Guide*.

Example

The following example sets up two BROWSEs with different indexes in different sessions:

```
SET EXCLUSIVE Off
CLOSE ALL
USE Company ORDER Company
BROWSE
CREATE SESSION
USE COMPANY ORDER CompCode
BROWSE
```

Portability

Not supported in dBASE IV or dBASE III PLUS.

See Also

DO, DEFINE, LOCAL, PRIVATE, PUBLIC, STATIC

CREATE VIEW

Table organization

Opens the Query Designer to create or modify a query file.

Syntax

CREATE VIEW <filename> | ? | <filename skeleton>

<filename> | ? | <filename skeleton> Specifies the name of the file you want to create. CREATE VIEW ? and CREATE VIEW <filename skeleton> display a dialog box, in which you can specify the name of a new query file. If you specify a file without including its path, *Visual* dBASE saves the file to the current drive and directory. If you specify a file without including an extension, *Visual* dBASE assumes a .QBE extension.

Description

CREATE VIEW performs the same operation as CREATE QUERY. Use either command to invoke the Query Designer, an interactive environment in which you can create or modify a query. The Query Designer creates a .QBE file that displays only the fields and records that meet specified conditions. To activate a query (.QBE) file, you can use the SET VIEW command. For more information on using the Query Designer, see the *User's Guide*.

Example

CREATE VIEW or CREATE QUERY with no arguments opens the query designer and generates an untitled query which the user will name later:

```
CREATE VIEW
```

CREATE VIEW followed by a file name will create a file of that name when the file is saved.

```
CREATE FILE Myquery
```

If MYQUERY.QBE already exists, you will be asked if you wish to overwrite it.

See Also

SET VIEW

CREATE VIEW...FROM ENVIRONMENT

Table organization

Creates a dBASE III PLUS-compatible view file based on the current working environment.

Syntax

CREATE VIEW <filename> FROM ENVIRONMENT

<filename> The view file to contain the current working environment specifications. By default, *Visual* dBASE assigns a .VUE extension to <filename> and saves the file in the current directory.

Description

This command saves the current working environment to a view (.VUE) file; you can open the file later with the SET VIEW TO <filename> command to restore the captured environment settings. The current working environment includes all open tables and index files, as well as their work area numbers, all relations, the active fields list, filter conditions, and any open format file. It does not, however, include SET RELATION...INTEGRITY or SET KEY options.

Example

The following example opens two tables, sets a relationship, filters, and field statements. It then issues CREATE VIEW which creates the .VUE file. CRT_VFE.VUE contains all these statements. SET VIEW TO reproduces them:

```
CLOSE DATABASE
USE Company EXCLUSIVE
SELECT 2
USE CONTACT EXCLUSIVE
INDEX ON Compcode TAG Compcode
SET FIELDS TO Contact
SELECT Company
SET RELATION TO Compcode INTO Contact
SET FIELDS TO Company, State_Prov
SET FILTER TO State_Prov="CA"
CREATE VIEW Crt_vfe FROM ENVIRONMENT
```

All these commands are reproduced when the user issues:

```
SET VIEW TO Crt_vfe
```

See Also

CREATE QUERY, CREATE VIEW, SET FIELDS, SET FILTER, SET FORMAT, SET INDEX, SET RELATION, SET VIEW, USE

CREATE...FROM

Table basics

Creates a table with the structure defined by using the COPY TO...STRUCTURE EXTENDED or CREATE...STRUCTURE EXTENDED commands.

Syntax

```
CREATE <filename 1> | ? | <filename skeleton 1>
[[TYPE] PARADOX | DBASE]
FROM <filename 2> | ? | <filename skeleton 2>
[[TYPE] PARADOX | DBASE]
```

<filename 1> | ? | <filename skeleton 1> The name of the table you want to create. CREATE ? and CREATE <filename skeleton> display a dialog box, in which you can specify the name of the destination table. If you specify a table without including its path, Visual dBASE saves the table to the current drive and directory. If you specify a table name without including an extension, defining a default table type with SET DBTYPE, or using one of the TYPE options, Visual dBASE assigns a .DBF extension. If you don't

specify a table name, the table remains untitled until you save it. If you specify an existing table name, *Visual dBASE* displays a dialog box, in which you can indicate whether you want to modify the existing table.

You can also create a table in a database (defined using the BDE Configuration Utility) by specifying the database as a prefix (enclosed in colons) to the name of the table, that is, *:database name:table name*. If the database is not already open, *Visual dBASE* displays a dialog box in which you specify the parameters, such as a login name and password, necessary to establish a connection to that database.

[TYPE] PARADOX | DBASE Specifies the type of table to create. The TYPE keyword is included for readability only; it has no effect on the operation of the command.

Specifying PARADOX creates a Paradox table with the .DB extension.

Specifying DBASE creates a dBASE table (the default). If you don't include an extension for *<filename>*, *Visual dBASE* assigns a .DBF extension.

FROM <filename 2> | ? | <filename skeleton 2>

[TYPE] PARADOX | DBASE Identifies the table whose structure you want to store in a new table. FROM ? and FROM *<filename skeleton>* display a dialog box, from which you can select the name of an existing table. If you specify a table without including its path, *Visual dBASE* looks for the table in the current directory, then in the path you specify with SET PATH. The Type and Database options are the same as those described in the previous paragraph.

You can also copy the structure of a table in a database (defined using the BDE Configuration Utility) by specifying the database as a prefix (enclosed in colons) to the name of the table, that is, *:database name:table name*. If the database is not already open, *Visual dBASE* displays a dialog box, in which you specify the parameters, such as a login name and password, necessary to establish a connection to that database.

Description

The CREATE...FROM command is most often used with the COPY TO...STRUCTURE EXTENDED command in a program to create a new table from another table that defines its structure, instead of using the interactive CREATE or MODIFY STRUCTURE commands. To do this, you can

- 1 Use COPY TO...STRUCTURE EXTENDED to create a table whose records provide information on each field of the original table.
- 2 Optionally, modify the structural data in the new table with any dBASE command used to manipulate data, such as REPLACE.
- 3 Use CREATE...FROM to create a new table from the structural information in the structure extended file. The new table is active when you exit CREATE...FROM.

The table created with CREATE...FROM becomes the current table in the currently selected work area. If the CREATE...FROM operation fails for any reason, no table remains open in the current work area.

If any fields in the table created with COPY TO...STRUCTURE EXTENDED have index flag fields set, CREATE...FROM also creates a production .MDX file with the specified index tags.

Example

The following example uses COPY TO ... STRUCTURE EXTENDED to create a temporary table containing the structure of Clients table that can be modified. CREATE...FROM is then used to create a new table with an altered structure:

```
SET SAFETY OFF
USE Clients
COPY TO Clients2 STRUCTURE EXTENDED
* Clients2.DBF now contains records that
* define Clients' structure.
USE Clients2 EXCLUSIVE
REPLACE Field_Name WITH "TELEPHONE" FOR RECNO()=9
CREATE NewClnr FROM Clients2
APPEND
CLOSE ALL
```

See COPY TO ... STRUCTURE EXTENDED for an additional example of using CREATE ... FROM.

See Also

COPY STRUCTURE, COPY TO...STRUCTURE EXTENDED, CREATE, DISPLAY STRUCTURE, LIST STRUCTURE, MODIFY STRUCTURE

CREATE...STRUCTURE EXTENDED

Table basics

Creates and opens a table that you can use to design the structure of a new table.

Syntax

```
CREATE <tablename> | ?
STRUCTURE EXTENDED
[[TYPE] PARADOX | DBASE]
```

<tablename> | ? The name of the table you want to create. CREATE ? STRUCTURE EXTENDED displays a dialog box, in which you can specify the name of the destination table. If you specify a table without including its path, *Visual dBASE* saves the table to the current drive and directory. If you specify a table name without including an extension, defining a default table type with SET DBTYPE, or using one of the TYPE options, *Visual dBASE* assigns a .DBF extension.

You can also create a table in a database (defined using the BDE Configuration Utility) by specifying the database as a prefix (enclosed in colons) to the name of the table, that is, *:database name:table name*. If the database is not already open, *Visual dBASE* displays a dialog box in which you specify the parameters, such as a login name and password, necessary to establish a connection to that database.

[TYPE] PARADOX | DBASE Specifies the type of table to create. The TYPE keyword is included for readability only; it has no effect on the operation of the command.

Specifying PARADOX creates a Paradox table with a .DB extension.

Specifying DBASE creates a dBASE table (the default). If you don't include an extension for <filename>, *Visual dBASE* assigns a .DBF extension.

Description

CREATE...STRUCTURE EXTENDED creates an empty table, called a *structure-extended table*, containing five fields of fixed names, types, and lengths. The fields correspond to attributes that describe fields in the table you want to create:

Field	Contents
FIELD_NAME	Character field that contains the name of the field.
FIELD_TYPE	Character field that contains the field's data type.
FIELD_LEN	Numeric field that contains the field length.
FIELD_DEC	Numeric field that contains the number of decimal places for numeric and float data.
FIELD_IDX	Character field (not created in dBASE III PLUS) that indicates if index tags were created on particular fields when the current table was created.

The CREATE...STRUCTURE EXTENDED command is similar to the COPY TO...STRUCTURE EXTENDED command. However, unlike COPY TO...STRUCTURE EXTENDED, which creates a table with records providing information on fields in the current table, CREATE...STRUCTURE creates an empty table. After using CREATE...STRUCTURE EXTENDED to create a new table, add records to define the structure of a new table. Then use the CREATE...FROM command to create a new table from the field definitions stored in the structure-extended table.

Example

The following example shows how CREATE ... STRUCTURE EXTENDED is used to create a table with fields representing table design values. After creating an empty file, the example appends blank records and places structure values within the fields. DBFrame is then used to create a new dBASE table called Names:

```
SET SAFETY OFF
CLOSE ALL
CLEAR ALL
CREATE DBFrame STRUCTURE EXTENDED
APPEND BLANK
REPLACE Field_Name WITH "FNAME"
REPLACE Field_Type WITH "C"
REPLACE Field_Len WITH 15
REPLACE Field_IDX WITH "N"
APPEND BLANK
REPLACE Field_Name WITH "LNAME"
REPLACE Field_Type WITH "C"
REPLACE Field_Len WITH 15
REPLACE Field_IDX WITH "N"
APPEND BLANK
REPLACE Field_Name WITH "ADDRESS"
REPLACE Field_Type WITH "C"
REPLACE Field_Len WITH 20
REPLACE Field_IDX WITH "N"
CREATE Names FROM DBFRAME TYPE DBASE
APPEND
```


Portability

Not supported in dBASE IV or dBASE III PLUS.

See Also

COPY TO...STRUCTURE EXTENDED, CREATE, CREATE...FROM

C

CTOD()**Expressions and type conversion**

Returns a specified character expression as a date expression.

Syntax

CTOD(<expC>)

<expC> The character expression, in the current date format, to return as a date expression.

Description

Use CTOD() to convert a character expression to a date expression. Once you convert character data to date data, you can manipulate it with date functions and date arithmetic.

You must enter <expC> in the current date format as determined by SET DATE, DBASEWIN.INI, or the Windows Control Panel (in that order). That is, SET DATE settings override those in DBASEWIN.INI, and settings in DBASEWIN.INI override those in the Windows Control Panel. Be sure <expC> matches the date format in use when your program runs.

CTOD() returns a date in the current date format.

If you pass an invalid date to CTOD(), it converts the date to a valid one and returns that date as a date expression. If you pass an empty or non-date string to CTOD(), it returns an empty date expression in the current format. For example, if the SET DATE format is AMERICAN, CTOD("") returns / /.

You can substitute the separator in <expC> with any character except a number. For example, <expC> can be "MM*DD*YY" or "MM!DD!YY". CTOD() returns the date with the current default separator.

You can specify a B.C. date by typing "bc" in uppercase or lowercase after the date and a space—for example, CTOD("4/4/92 BC").

Example

The following examples uses CTOD() to change character data to a date format:

```
SET DATE AMERICAN
? CTOD("4/1/94")           && Returns 04/01/94
? CTOD("14/1/94")          && Returns 01/01/94
? CTOD("4/32/94")          && Returns 05/02/94
? CTOD("00/00/00")          && Returns / /
? CTOD("")                  && Returns / /
```

DATABASE()

```
? CTOD("X/X/X")           && Returns   /   /
? CTOD(4/1/94)             && Returns an error
x = "4/1/94"
? CTOD(x)                  && Returns 04/01/94
? CTOD("1/1/91 BC")        && Returns 01/01/91 BC
```

See Also

DTOC(), DTOS(), SET DATE, SET CENTURY, SET MARK

DATABASE()

Table basics

Returns the name of the current database from which tables are accessed.

Syntax

DATABASE()

Description

DATABASE() returns the name of the current default database selected with the SET DATABASE command. (Databases are defined with the BDE Configuration Utility; see *Getting Started* for more information.) If no database is open, the DATABASE() function returns an empty string ("").

Example

The following example uses OPEN DATABASE and SET DATABASE TO to connect to databases on a database server and DATABASE() to confirm which database is open:

```
CLEAR
SET DBTYPE TO                && Default is DBASE
OPEN DATABASE CAClients ;    && Establish connection with database server;
    LOGIN guest/guest        and enter user name / password

OPEN DATABASE FLClients
    LOGIN guest/guest        && Establish connection with database server;
                                and enter user name / password
SET DATABASE TO CAClients    && Makes it current database
USE admin.Oakland
? DATABASE()                 && Returns CAClients
? ALIAS()                    && Returns 1
? DBF()                      && Returns ADMIN.OAKLAND
? WORKAREA()                 && Returns 1
SET DATABASE TO FLClients    && Makes it Current database
? DATABASE()                 && Returns FLClients
CLOSE DATABASES CAClients, FLClients ; && Disconnect from database server
```

Portability

Not supported in dBASE IV or in dBASE III PLUS.

See Also

CLOSE..., OPEN DATABASE, SET DATABASE, SET DBTYPE

DATE()

Date and time data

Returns the system date.

Syntax

DATE()

D

Description

DATE() returns a date expression that is your computer system's current date.

dBASE uses the value set by SET DATE, DBASEWIN.INI, or the International option of the Windows Control Panel (in that order) to determine the current date format. That is, SET DATE settings override those in DBASEWIN.INI, and settings in DBASEWIN.INI override those in the Windows Control Panel. DATE() returns a date in the current date format.

If SET CENTURY is ON, DATE() returns the date with a 4-digit year.

To change the system date, use SET DATE TO. To express the date as a character string, use DTOC() or DTOS().

Example

The following example uses DATE() to compute the number of days between a date field's contents and the current system date, and to derive a due date 30 days in the future.

```
USE Clients
? DTOC(Baldate) + ", " + ;
  LTRIM(STR(DATE() - Baldate)) + ;
  " days have passed since this account was balanced."
CLOSE DATABASE
?
? "Your account is due in thirty (30) days - "
?? DATE() + 30
```

See Also

DTOC(), DTOS(), SET CENTURY, SET DATE, SET DATE TO, SET MARK

DAY()

Date and time data

Returns the numeric value of the day of the month for a specified date expression.

Syntax

DAY(<expD>)

<expD> The date expression whose corresponding day-of-the-month number to return.

Description

DAY() returns a date's day of the month number—a value from 1 to 31.

DBERROR()

Enter *<expD>* in the current date format as determined by SET DATE, DBASEWIN.INI, or the International option of the Windows Control Panel (in that order). That is, SET DATE settings override those in DBASEWIN.INI, and settings in DBASEWIN.INI override those in the Windows Control Panel. Be sure *<expD>* matches the date format in use when your program runs.

If you pass an invalid date to DAY(), dBASE converts the date to a valid one and returns the month-day number of that date. If you pass an empty or non-date expression delimited with braces ({ }) to DAY(), it returns 0. If you pass a non-date expression or an expression that isn't delimited with braces to DAY(), it returns an error.

Example

The following examples use DAY() to return the numeric day of the month from date type data.

```
SET DATE AMERICAN
? DAY({4/1/94})           && Returns 1
? DAY({4/32/93})          && Returns 2 because April has 30 days
? DAY({00/00/00})          && Returns 0
? DAY({})                 && Returns 0
? DAY(X)                  && Returns error message
? DAY(4/1/94)             && Returns error message
X = {4/1/94}
? DAY(X)                  && Returns 1
```

For additional examples of DAY() see CMONTH() and CDOW().

See Also

DOW(), MONTH(), SET CENTURY, SET DATE

DBERROR()

Error handling and debugging

Returns the number of the last IDAPI error.

Syntax

DBERROR()

Description

DBERROR() returns the IDAPI error number of the last IDAPI error generated by the current table. To learn the IDAPI error message itself, use DBMESSAGE().

See the table in the description of ERROR() that compares ERROR(), MESSAGE(), DBERROR(), DBMESSAGE(), SQLERROR(), SQLMESSAGE(), and CERROR().

See online Help for a listing of all error messages.

Example

The following example uses ON ERROR to branch to an error procedure that uses DBERROR() to return what IDAPI error has occurred during the BROWSE and DBMESSAGE() to return what it means:

```
USE Clients
ON ERROR DO Recovery
COPY TO TEMP
USE TEMP
BROWSE

PROCEDURE Recovery
CLOSE DATABASES
CLEAR
IF ERROR()=239
? "The IDAPI error was error number: " + STR(DBERROR())
? "Which means: " + DBMESSAGE()
ELSE
? "No IDAPI error encountered"
ENDIF
RETURN
```

D**Portability**

Not supported in dBASE IV or dBASE III PLUS.

See Also

CERROR(), DBMESSAGE(), ERROR(), MESSAGE(), SQLERROR(), SQLMESSAGE()

DBF()

Table basics

Returns the name of a table open in the current or a specified work area.

Syntax

DBF([<alias>])

<alias> A work area number (1 through 225), letter (A through J), or alias name. The work area letter or alias name must be enclosed in quotes.

Description

DBF() returns the name of the table open in a specified work area. If SET FULLPATH is ON, the DBF() function also returns the drive and directory location of the table in addition to the table name. If you do not specify a work area, the current work area is assumed.

If no table is in use in the current or specified work area, DBF() returns an empty string ("").

DBMESSAGE()

Example

The following example demonstrates the relationship between DBF(), SELECT() and WORKAREA() when multiple tables are open in more than one work area:

```
CLOSE ALL
USE Flights IN 1
USE Aircrdb IN 2
USE Company IN 3
SELECT 3
? DBF()           && Returns C:COMPANY.DBF
? SELECT()        && Returns 4
? WORKAREA()      && Returns 3
```

See Also

ALIAS(), MDX(), NDX(), SET FULLPATH, TAG(), WORKAREA(), USE

DBMESSAGE()

Error handling and debugging

Returns the error message of the last IDAPI error.

Syntax

DBMESSAGE()

Description

DBMESSAGE() returns the error message of the most recent IDAPI error.

See the table in the description of ERROR() that compares ERROR(), MESSAGE(), DBERROR(), DBMESSAGE(), CERROR(), SQLERROR(), and SQLMESSAGE().

See online Help for a listing of all error messages.

Example

See DBERROR() for an example of using DBMESSAGE().

Portability

Not supported in dBASE IV or dBASE III PLUS.

See Also

CERROR(), DBERROR(), ERROR(), MESSAGE(), SQLERROR(), SQLMESSAGE()

DEACTIVATE MENU

dBASE IV menus

Erases and disables an active dBASE IV menu bar without removing its definition from memory. This command is supported primarily for compatibility with dBASE IV. In *Visual* dBASE, use RELEASE OBJECT to clear an object from a form.

For complete syntax information on DEACTIVATE MENU, see online Help. For information about defining forms, see the Forms chapters in the *User's Guide*.

DEACTIVATE POPUP

dBASE IV menus

D

Erases and disables an active dBASE IV pop-up menu without removing its definition from memory. This command is supported primarily for compatibility with dBASE IV. In *Visual* dBASE, use RELEASE OBJECT to clear an object from a form.

For complete syntax information on DEACTIVATE POPUP, see online Help. For information about defining forms, see the Forms chapters in the *User's Guide*.

DEACTIVATE WINDOW

dBASE IV windows

Clears from the screen windows that were displayed and enabled with the dBASE IV command ACTIVATE WINDOW, without releasing their definitions from memory. This command is supported primarily for compatibility with dBASE IV. In *Visual* dBASE, use CLOSE FORMS or RELEASE OBJECT to close or release a form.

For complete syntax information on DEACTIVATE WINDOW, see online Help. For information about defining forms, see the Forms chapters in the *User's Guide*.

DEBUG

Error handling and debugging

Opens the dBASE Debugger.

Syntax

DEBUG

```
[<filename> | ? | <filename skeleton> |  
<procedure name> | <UDF name>  
[WITH <parameter list>]]
```

<filename> | ? | <filename skeleton> The file to debug. DEBUG ? and DEBUG <filename skeleton> display the Open Source File dialog box, from which you can select a file. If you specify a file without including its path, dBASE looks for the file in the current directory, then in the path you specify with SET PATH. If you specify a file without including its extension, dBASE assumes .PRG.

<procedure name> | <UDF name> The procedure or user-defined function (UDF) in an open program file to debug. The procedure or UDF must be in the program file containing the DEBUG command that calls it, or in a separate open procedure file on the *search path*. (You can open a procedure file with SET PROCEDURE TO.) See the description of DO for an explanation of the search path and order dBASE follows when it encounters a call to a program, procedure, or UDF.

WITH <parameter list> Specifies expressions to pass as parameters to a program, procedure, or UDF that contains PARAMETERS <parameter list> as its first executable

command line. For information about parameter passing, see the description of PARAMETERS.

You can debug a UDF without DEBUG...WITH by calling the UDF directly in an expression; in which case, you enclose parameters to pass to it in parentheses after the UDF name. For more information, see the last paragraph of the next section.

If you want to specify a parameter list, you must also specify one of the preceding options. That is, issuing DEBUG WITH *<parameter list>* returns an error.

Description

Use DEBUG to turn on the Debugger and view or control program execution interactively. The Debugger displays information about the state of a program, procedure, or UDF—and any procedure files that the program or subroutine opens—during execution. For more information, see the *Programmer's Guide*, which describes the Debugger in detail.

You can issue DEBUG without any options from the Command window or from within a program file. If you issue DEBUG without any options in the Command window, dBASE opens the Debugger without loading a program or subroutine. (You can load a file to debug from the Debugger.) If you issue DEBUG without any options in a program file, dBASE loads the current program file in the Debugger. If you issue DEBUG *<filename>* from a program file, *<filename>* should be a program file *other* than the one containing the command; otherwise, you get an error.

You can issue DEBUG *<procedure name>* and DEBUG *<UDF name>* from within the program file that calls the procedure or UDF. You can also issue both of these commands from the Command window or from within a program file other than the one containing the procedure or UDF, but the file must be on the search path.

You can run a program line by line in the Debugger and control which lines to stop at. If you don't want to stop execution within a subroutine, you can step over the command line that calls the subroutine. The call to the subroutine is still made, and the subroutine still executes; you just don't see the line-by-line execution of the subroutine. The Debugger stops at the first command line following the subroutine.

When you debug a UDF with DEBUG *<UDF name>* WITH *<parameter list>*, dBASE ignores the return value of the UDF. If you want to call a UDF in the context of an expression so that you can see if the UDF returns the right value and the right data type, you can, in a program file, issue DEBUG with no parameters and then the command-line expression containing the call to the UDF. DEBUG will open the program file in the Debugger, in which you can then step to the next line containing the UDF call.

In the Debugger, step over a command line that calls a subroutine by doing one of the following:

- Click the Step Over icon on the Debugger SpeedBar
- Choose Run | Step Over from the Debugger menu
- Press *F8*

Example

The following examples demonstrate three options for calling the Debugger. Type in the Command window:

```
DEBUG ?      && Opens "Open Source File" dialog box
DEBUG *.WFM  && Opens "Open Source File" dialog box with .WFM files selectable
DEBUG ANIMALS && Enters Debugger with Animals.PRG source code in the upper-left window
```

D

Including the following code line in your program automatically takes you to the Debugger. SET ECHO ON is another way to accomplish the same thing:

```
ON ERROR DEBUG PROGRAM()
```

See Also

DISPLAY COVERAGE, GENERATE, ON ERROR, RESUME, SET COVERAGE, SET PATH, SUSPEND

DECLARE

Memory variables

Defines one or more fixed arrays.

Syntax

```
DECLARE <array name 1>["<expN list 1>"]
[,<array name 2>["<expN list 2>"]...]
```

Brackets ([]) in quotation marks are required syntax components.

<array name 1>[,<array name 2>...] The memory variable(s) that are the name(s) of the array(s).

["<expN list 1>"][,...["<expN list 2>"][,...] Numeric or float expressions (from 1 to 254 inclusive). The number of expressions you specify determines the number of dimensions of the array. Each one of the expressions specifies how many values (data elements) that dimension has. For example, if [<expN list 1>] is [3,4], dBASE defines a two-dimensional array with three rows and four columns.

Description

Use DECLARE to define an array of a specified size as a memory variable. Array elements can be of any data type. (An array element can also specify the name of another array.) A single array can contain multiple data types. When you use DECLARE, all array elements are initialized to a logical data type with a value of .F.

The array can hold as many elements as memory allows. You can create arrays that contain more than two dimensions, but most dBASE array functions work only on one- or two-dimensional arrays.

There are two ways to refer to individual elements in an array; you can use either the element *subscripts* or the element *number*. Element subscripts indicate the row and column in which an element is located. Element numbers indicate the sequential position of the element in the array, starting at the first row and first column of the array. To determine the number of elements, rows, or columns in an array, use ALEN().

DECLARE

Certain dBASE functions require the element number, and others require the subscripts. If you are using one- or two-dimensional arrays, you can use AELEMENT() to determine the element number if you know the subscripts, and ASUBSCRIPT() to determine the subscripts if you know the element number.

After you create an array, you can place values in cells of the array using STORE, or you can use =. You can also use AFILL() to place the same value in a range of cells in the array. To add or delete elements from an array, use ADEL() and AINS(). To resize an array, or make a one-dimensional array two-dimensional, use AGROW() or ARESIZE(). For more information on using arrays, see Chapter 5 in the *Programmer's Guide*.

You can pass array elements as parameters, and you can pass a complete array as a parameter to a program or procedure by specifying the array name without a subscript. For more information on passing arrays as parameters, see Chapter 4 in the *Programmer's Guide*.

Example

The following example relates two tables to create a view consisting of two fields from each table, then uses DECLARE to create an array Compsum and copies the selected data to the array. The counting DO WHILE loop displays the contents of the array to the results pane of the Command window:

```
CLOSE ALL
CLEAR
USE Contact IN SELECT() ORDER CompCode
USE Company IN SELECT()
SELECT Company
SET RELATION TO CompCode INTO Contact

DECLARE Compsum[5,4]                && Create an array of 5 rows and 4 columns
COPY TO ARRAY Compsum NEXT 5;
    FIELDS Company->Company, ;
    Company->City, Contact->CompCode, ;
    Contact->Contact
Cnt=1
DO WHILE Cnt<=5
    ? Compsum[Cnt,1], Compsum[Cnt,2]
    ?? Compsum[Cnt,3], Compsum[Cnt,4]
    Cnt=Cnt+1
ENDDO
CLOSE ALL
```

Portability

Not supported in dBASE III PLUS.

See Also

ADEL(), AELEMENT(), AFILL(), AGROW(), AINS(), ALLEN(), APPEND FROM ARRAY, ARESIZE(), ASUBSCRIPT(), COPY TO ARRAY, REPLACE FROM ARRAY, STORE

DEFINE

Creates an object from a class.

Syntax

```
DEFINE <class name> <object name>
[OF <container object>]
[FROM <row, col> TO <row, col> > | <AT <row, col>]
[PROPERTY <stock property list>]
[CUSTOM <custom property list>]
[WITH <parameter list>]
```

<class name> The class of the object you create. DEFINE can create objects from twenty standard classes:

Browse	Checkbox	Combobox
DDELink	DDETopic	Editor
Entryfield	Form	Image
Line	Listbox	Menu
Object	Ole	Pushbutton
Radiobutton	Rectangle	Scrollbar
Spinbox	Text	

You can also specify a custom class, which you define with the CLASS...ENDCLASS command.

<object name> The identifier for the object you create. *<object name>* is the object reference variable, through which you can access and change the object properties. For information on using object reference variables, see Chapter 10 in the *Programmer's Guide*.

OF <container object> Identifies the object that contains the object you define. (For most UI objects the container object is a form.) OF *<container object>* is required if you identify the object using *<index operator expN>*.

FROM <row>, <col> TO <row>, <col> | AT <row>, <col> Specifies the initial location and size of the object within its parent form. FROM and TO specify the upper left and lower right coordinates of the object, respectively. AT specifies the position of the upper left corner.

PROPERTY <stock property list> Specifies values you assign to the built-in properties of the object.

CUSTOM <custom property list> Specifies new properties you create for the object and the values you assign to them. For information on custom properties, see Chapter 10 in the *Programmer's Guide*.

WITH <parameter list> Specifies the parameters you pass to the object. Declare these parameters with the PARAMETERS clause of the CLASS...ENDCLASS command.

Description

Use DEFINE to create an object definition in memory.

An object contains memory variables called *properties*. Some properties hold data values that modify the object itself, while others reference subroutines that are executed in response to events. For example, a pushbutton responds to a mouse click by triggering its OnClick subroutine and the OnSelection subroutine of its parent form.

Every object belongs to a class. A class is a specification, or template, for a type of object. Visual dBASE provides many built-in classes that you can use to create common Windows objects (also called controls), such as radio buttons, pushbuttons, and entry fields. For example, each time you create a form with DEFINE FORM, you make a new form object that has the built-in specifications from the Form class.

You can design your own *custom class* with the CLASS...ENDCLASS command. CLASS...ENDCLASS lets you use regular dBASE code to declare the properties and methods that objects of the class will have. For information on classes, see Chapter 11 in the *Programmer's Guide*.

Example

The following example uses DEFINE to create a custom DIALOG BOX class, with several useful functions:

```

rDialog = NEW Dialog()
rDialog.alert("This is an alert box")
? rDialog.YesNo("Do you like it?")
? rDialog.GetString("What is your name?";,
    SPACE(20),"@!", "Last Name first")
RETURN

CLASS Dialog
    this.Top = 20
    this.Left = 12
    this.Height = 7
    this.Width = 12
    this.Value = .f.
    this.Color = 'n/w'
    SET CUAENTER OFF

FUNCTION Alert
PARAMETER cString, cTitle
PRIVATE nButtonCol
    this.Height = 8
    this.Value = ""
    this.Left = 39 - (this.Width/2)
    IF TYPE( 'cTitle') <> 'C'
        this.Text = ""
        this.Height = 7
    ELSE
        this.Text = cTitle
        this.Height = 9
    ENDIF
    this.Width = MAX(LEN(cString),LEN(this.Text))+7
DEFINE FORM F1;
```

```

PROPERTY Top this.Top,;
    Left this.Left, ;
    Height this.Height, ;
    Width this.Width, ;
    MDI .F., ;
    Text this.Text
DEFINE TEXT T1 of F1;
    PROPERTY Top 1, Left 3, ;
        Width LEN(cString), ;
        Text cString
        nButtonCol = (this.width/2)-5
    DEFINE PUSHBUTTON B1 of F1;
        PROPERTY Top 4, Left nButtonCol, ;
            Text "OK", OnClick {; Form.CLOSE()}
F1.READMODAL()
F1.RELEASE()
RETURN .T.

FUNCTION GetString
PARAMETER cPrompt, cString, cPicture, cTitle
RETURN this.GetValue(cPrompt,cString,LEN(cString),;
    cPicture, cTitle)

FUNCTION GetNumber
PARAMETER cPrompt, nNumber, cPicture, cTitle
RETURN this.GetValue(cPrompt,nNumber,;
    LEN(STR(nNumber)), cPicture, cTitle )

FUNCTION GetDate
PARAMETER cPrompt, dDate, cPicture, cTitle
RETURN CTOD(this.GetValue(cPrompt,DTOC(dDate),8,;
    cPicture, cTitle))

FUNCTION YesNo
PARAMETER cString, cTitle, lInitValue
PRIVATE nButtonCol1, lRetval
    this.Height = 8
    lRetval = lInitValue
    this.Value = .t.
    this.Left = 39 - (this.width/2)
    IF TYPE( 'cTitle' ) <> 'C'
        this.Text = ""
        this.Height = 7
    ELSE
        this.Text = cTitle
        this.Height = 9
    ENDIF
    this.Width=MAX(18,MAX(LEN(cString),;
        LEN(this.Text))+ 7 )
DEFINE FORM F1;
    PROPERTY Top this.Top, ;
        Left this.Left, ;
        Height this.Height, ;
        Width this.Width, ;
        MDI .F., ;

```

DEFINE

```
Text this.Text
DEFINE TEXT T1 OF F1;
  PROPERTY Top 1, left 3, ;
    Width LEN(cString), ;
    Text cString
    nButtonCol1 = (this.width/2) - 9
  DEFINE PUSHBUTTON Yes OF F1;
    PROPERTY Top 4, Left nButtonCol1, ;
      Width 5, Text "Yes", OnClick this.SetYes
  DEFINE PUSHBUTTON No OF F1;
    PROPERTY Top 4, Left nButtonCol1 + 12,;
      Width 5, Text "No", OnClick this.SetNo
F1.READMODAL()
F1.RELEASE()
RETURN lRetVal

FUNCTION GetValue
PARAMETER cPrompt, cString, nLen, cPicture, cTitle
PRIVATE nButtonCol
  this.Width = nLen + 8 + LEN(cPrompt)
  this.Value = ""
  this.Left = 39 - (this.width/2)
  IF TYPE( 'cTitle' ) <> 'C'
    this.Text = ""
    this.Height = 8
  ELSE
    this.Text = cTitle
    this.Height = 10
  ENDIF
  this.Width=MAX(LEN(cString)+nLen,;
    LEN(this.Text)) + 8
DEFINE FORM F1;
  PROPERTY Top this.Top, ;
    Left this.Left, ;
    Height this.Height, ;
    Width this.Width, ;
    MDI .F., ;
    Text this.Text, ;
    ColorNormal this.Color

  DEFINE TEXT T1 OF F1;
    PROPERTY Top 1, Left 3, ;
      WIDTH LEN(cPrompt), ;
      Text cPrompt
      IF TYPE('cPicture') = 'C'
        DEFINE ENTRYFIELD E1 OF F1;
          PROPERTY Top 1, Left 4+LEN(cPrompt),;
            Width nLen, ;
            Value cString, ;
            Datalink "cString", ;
            Picture cPicture
      ELSE
        DEFINE ENTRYFIELD E1 OF F1;
          PROPERTY Top 1, Left 4 + LEN(cPrompt), ;
            Width nLen, ;
```

```

        Value cString, ;
        Datalink "cString"
    ENDIF
    nButtonCol = (this.width/2) - 5
    DEFINE PUSHBUTTON B1 OF F1;
        PROPERTY Top 4, Left nButtonCol, ;
        Text "OK", OnClick {; Form.Close()}
F1.READMODAL()
F1.RELEASE()
RETURN cString

FUNCTION SetYes
lRetval = .T.
Form.CLOSE()
RETURN .T.

FUNCTION SetNo
lRetval = .F.
Form.CLOSE()
RETURN .T.
ENDCLASS

```

D

Portability

Not supported in dBASE IV or dBASE III PLUS.

See Also

CLASS...ENDCLASS, REDEFINE

DEFINE BAR

dBASE IV menus

Creates a bar and its prompt in a dBASE IV pop-up menu. This command is supported primarily for compatibility with dBASE IV. In *Visual dBASE*, use DEFINE, OPEN FORM, and READMODAL() to create and activate menus associated with forms.

For complete syntax information on DEFINE BAR, see online Help. For information about defining forms, see the Forms chapters in the *User's Guide*.

DEFINE BOX

Printing

Defines a character-mode box to draw around ? command output. This command is supported primarily for compatibility with dBASE IV. In *Visual dBASE*, use DEFINE to create and place rectangles on forms.

For complete syntax information on DEFINE BOX, see online Help. For information about creating rectangles and forms, see the Forms chapters in the *User's Guide*.

Creates and names a customized color.

Syntax

```
DEFINE COLOR <color name>
<red expN>, <green expN>, <blue expN>
```

<red expN>, <green expN>, <blue expN> Specifies the proportions of red, green, and blue that make up the defined color. Each number determines the intensity of the color it represents, and can range from 0 (least intensity) to 255 (greatest intensity).

Description

Use DEFINE COLOR to create a customized color combination. Once you have defined <color name>, you can use it instead of one of the standard colors such as R, W, BG, and so on.

The color you create with DEFINE COLOR is based on three numbers, <red expN>, <green expN>, and <blue expN>. Adjusting these numbers alters the color you create. For example, increasing or decreasing <green expN> increases or decreases the amount of green contained in the customized color.

Use the GETCOLOR() function to open a dialog box in which you create a custom color or choose from a palette of available colors. After exiting GETCOLOR(), issue DEFINE COLOR with the values it returns to define the desired color.

You can't use standard color names (such as blue, red, white or black) as <color name>. If you do, dBASE returns an error when you later issue SET COLOR TO <color name>. Also, you can't redefine a dBASE color with a command such as DEFINE COLOR R <red expN> , <green expN> , <blue expN>. If you later use R in a SET COLOR TO statement, dBASE uses its internal value for R, which is red, rather than the color you defined.

Example

The following example defines white, black, red, green, blue, and yellow using DEFINE COLOR and then uses them with SET COLOR TO:

```
Oldcolors=SET("ATTRIBUTE")
DEFINE COLOR Col_white 255, 255, 255
* Equivalent to W
DEFINE COLOR Col_black 0, 0, 0
* Equivalent to N
DEFINE COLOR Col_red 255, 0, 0
* Equivalent to R
DEFINE COLOR Col_green 0, 255, 0
* Equivalent to G
DEFINE COLOR Col_blue 0, 0, 255
* Equivalent to B
DEFINE COLOR Col_yellow 255, 255, 0
*
SET COLOR TO Col_white/Col_green
? "Now is the time"
```



```
?? "Col_white/Col_green" AT 40
SET COLOR TO Col_red/Col_blue
? " for all good"
?? "Col_red/Col_blue" AT 40
SET COLOR TO Col_blue/Col_red
? " men and women"
?? "Col_blue/Col_red" AT 40
SET COLOR TO Col_blue/Col_green
? " to come to "
?? "Col_blue/Col_green" AT 40
SET COLOR TO Col_red/Col_blue
? " as soon as"
?? "Col_red/Col_blue" AT 40
SET COLOR TO Col_yellow/Col_green
? " they possibly can"
?? "Col_yellow/Col_green" AT 40
WAIT
SET COLOR TO &Oldcolors && Reset colors
CLEAR
```

Portability

Not supported in dBASE IV or dBASE III PLUS.

See Also

ColorHighlight, ColorNormal, GETCOLOR(), SET COLOR TO, SET COLOR OF

DEFINE MENU

dBASE IV menus

Names a dBASE IV menu bar and begins its definition in memory. This command is supported primarily for compatibility with dBASE IV. In *Visual dBASE*, use DEFINE, OPEN FORM, and READMODAL() to create and activate menus associated with forms.

For complete syntax information on DEFINE MENU, see online Help. For information about defining forms, see the Forms chapters in the *User's Guide*.

DEFINE PAD

dBASE IV menus

Creates and names a pad in an existing dBASE IV menu bar. This command is supported primarily for compatibility with dBASE IV. In *Visual dBASE*, use DEFINE, OPEN FORM, and READMODAL() to create and activate menus associated with forms.

For complete syntax information on DEFINE PAD, see online Help. For information about defining forms, see the Forms chapters in the *User's Guide*.

DEFINE POPUP

dBASE IV menus

Creates a dBASE IV popup menu and stores the definition in memory. This command is supported primarily for compatibility with dBASE IV. In *Visual* dBASE, use DEFINE, OPEN FORM, and READMODAL() to create and activate menus associated with forms.

For complete syntax information on DEFINE POPUP, see online Help. For information about defining forms, see the Forms chapters in the *User's Guide*.

DEFINE WINDOW

dBASE IV windows

Creates a dBASE IV-style window, a rectangular area in which menus and input areas can be displayed and enabled. This command is supported primarily for compatibility with dBASE IV. In *Visual* dBASE, use DEFINE to create *forms*, which are used instead of dBASE IV windows.

For complete syntax information on DEFINE WINDOW, see online Help. For information about defining forms, see the Forms chapters in the *User's Guide*.

DELETE

Fields and records

For dBASE tables, marks records for deletion. When accessing Paradox or SQL tables, DELETE removes records from the table.

Syntax

DELETE

[<scope>]

[FOR <condition 1>]

[WHILE <condition 2>]

<scope> The number of records to delete. RECORD <*n*> identifies a single record by its record number. NEXT <*n*> identifies *n* records, beginning with the current record. ALL specifies all records. REST specifies all records from the current record to the end of the file.

FOR <condition 1>

WHILE <condition 2> Determines which records are affected by DELETE. FOR restricts DELETE to records that meet <condition 1>. WHILE starts processing with the current record and continues with each subsequent record as long as <condition 2> is true.

Description

When you use DELETE to mark records for deletion in a dBASE table, *Visual* dBASE doesn't actually remove them from the table. It removes records marked for deletion when you issue PACK. DELETE with no options marks only the current record. When using a Paradox or SQL table, the DELETE command removes records from the table immediately (similar to issuing a PACK command with a dBASE table).

The mark that appears next to deleted records in a list or display is an asterisk.

To work with a table as if you have issued PACK, but without deleting marked records irrevocably, turn SET DELETED ON (the default). To undelete records, issue the RECALL command. However, use ZAP to remove all records.

Example

The following example uses DELETE to mark all records with StartBal less than \$750 and SET DELETED ON to exclude those marked records from the subsequent BROWSE:

```
USE Clients
DELETE FOR StartBal < 750
SET DELETED ON
GO TOP
BROWSE FIELDS Company, City, State_Prov, Zip_P_Code,;
      StartBal NOMODIFY NOAPPEND NODELETE
RECALL ALL           && Removes deletion marks
CLOSE ALL
```

The following example uses DELETE to mark five randomly selected contest winners and list them to the screen. You can also use LABEL FORM in a similar way to print labels:

```
SET TALK OFF
SET DELETED OFF
USE Company
RECALL ALL
SET DECIMALS TO 0
Cnt=1
DO WHILE Cnt<=5
  STORE (INT(RAND()*RECCOUNT()+1) TO Mark
  GOTO Mark
  IF .NOT. DELETED()
    DELETE
    Cnt=Cnt+1
  ENDIF
ENDDO
SCAN FOR DELETED()
  ? Company
ENDSCAN
```

The same concept applies if you want to create a new table from these randomly selected records:

```
SET SAFETY OFF
COPY TO Winners FOR DELETED()
RECALL ALL
USE Winners
BROWSE
```

See Also

PACK, RECALL, SET DELETED, ZAP

DELETE FILE

Removes a file from a disk.

Syntax

DELETE FILE <filename> | ? | <filename skeleton>

<filename> | ? | <filename skeleton> Identifies the file to remove. ? and <filename skeleton> display a dialog box from which you can select a file.

If you specify a file without including its path, dBASE looks for the file in the current directory, then in the path you specify with SET PATH. If you specify a file without including its extension, dBASE does not assume an extension.

Description

DELETE FILE and ERASE are equivalent commands. See ERASE for more information.

Example

The following examples use DELETE FILE:

```
DELETE FILE Temp.prg
DELETE FILE ?
* Displays the open source dialog box
```

Portability

Not supported in dBASE III PLUS, but ERASE is. The <filename skeleton> argument is not supported in dBASE IV.

See Also

ERASE, RENAME, SET PATH

DELETE TABLE

Deletes a specified table.

Syntax

DELETE TABLE <table name> | ? | <filename skeleton>
[[TYPE] PARADOX | DBASE]

<table name> | ? | <filename skeleton 1> The name of the table that you want to delete. DELETE TABLE ? and DELETE <filename skeleton> display a dialog box, in which you can select the table you want to delete.

You can also delete a table in a database (defined using the BDE Configuration Utility) by specifying the database as a prefix (enclosed in colons) to the name of the table, that is, :database name:table name. If the database is not already open, Visual dBASE displays a dialog box in which you specify the parameters, such as a login name and password, necessary to establish a connection to that database.

[TYPE] PARADOX | DBASE Specifies the type of table to delete, which can include either Paradox or dBASE tables.

Description

Use the DELETE TABLE command to delete tables and associated .NDX and .MDX index files. Make sure the table is not in use before you attempt to delete it.

D

Example

The following example uses DELETE TABLE to delete two temporary tables after creating an adhoc view of Clients table:

```
SET SAFETY OFF
USE Clients
COPY TO NewNames STRUCTURE EXTENDED
USE NewNames EXCLUSIVE
BROWSE                && Delete unwanted fields or edit design from table
PACK                  && Saves new structure
CREATE NewCInt FROM NewNames
APPEND FROM Clients FOR State_Prov = "CA"
GO TOP
BROWSE
CLOSE DATABASES
DELETE TABLE NewNames TYPE DBASE
DELETE TABLE NewCInt TYPE DBASE
```

Portability

Not supported in dBASE IV or dBASE III PLUS.

See Also

DELETE FILE, DELETE TAG, ERASE

DELETE TAG

Table organization

Deletes specified index tags from .MDX files. You can also use this command to delete individual index tags defined for Paradox and SQL tables.

Syntax

```
DELETE TAG <tag name 1>
[OF <filename 1> | ? | <filename skeleton 1>]
[, <tag name 2>
[OF <filename 2> | ? | <filename skeleton 2>]...]
```

<tag name 1>, <tag name 2>, ... <tag name n> The index tag names to delete from .MDX files.

OF <filename 1> | ? | <filename skeleton 1> Specifies the .MDX file containing the tag name to delete. OF ? and OF <filename skeleton 1> display a dialog box, in which you select a multiple index file. If you specify a file without including its path, *Visual* dBASE looks for the file in the current directory, then in the path you specify with SET PATH. If you specify a file without including an extension, *Visual* dBASE assumes an .MDX extension.

DELETED()

If you don't specify an index file, *Visual* dBASE assumes the index tag you want to delete is in the index file with the same name as the current table.

Description

Use DELETE TAG to delete index tags from .MDX files for dBASE tables or secondary indexes on a Paradox table. *Visual* dBASE allows a maximum of 47 index tags in a single .MDX file, so deleting unneeded tags frees slots for new tags as well as reducing the amount of disk space and memory that an .MDX file requires.

For dBASE tables, the .MDX file must be open when you delete the tags. If you delete all tags in an .MDX file, the .MDX file is also deleted. If you delete the production .MDX file by deleting all index tags, the table file header is updated to indicate there is no longer a production index associated with the table.

In a multiuser environment, the table associated with the indexes you want to delete must be opened in exclusive mode. When accessing a Paradox table, specifying DELETE TAG without an argument deletes the primary index.

Example

The following example creates a temporary index, uses it in a BROWSE command then removes it with DELETE TAG:

```
USE Company EXCLUSIVE
INDEX ON Sic_code TAG Sic_code
BROWSE FIELDS Sic_code, Company
* now the Sic_code index is not needed, delete it
DELETE TAG Sic_code
```

Portability

Not supported in dBASE III PLUS. Also, the *<filename skeleton>* option is not available in dBASE IV.

See Also

CLOSE INDEXES, COPY INDEXES, SET INDEX, TAG()

DELETED()

Fields and records

Indicates if records in the current or a specified table have been marked for deletion.

Syntax

DELETED([*<alias>*])

<alias> A work area number (1 through 225), letter (A through J), or alias name. The work area letter or alias name must be enclosed in quotes.

Description

DELETED() returns .T. if the current record in the specified work area is marked for deletion; otherwise, DELETED() returns .F.. If you do not specify a work area, the current work area is assumed.

If no table is open in the current or specified work area, DELETED() also returns .F..

D**Example**

The following example uses DELETE to mark a subset of the total table. DELETED() is then used to perform a BROWSE that displays only that subset of records and to create another table (Cal) that contains only California records:

```
SET TALK OFF
SET SAFETY OFF
USE Clients
SET DELETED OFF
DELETE FOR State_Prov = "CA"
GO TOP
BROWSE FIELDS Company, City, State_Prov FOR DELETED()
COPY TO Cal FOR DELETED()
USE Cal
BROWSE
CLOSE ALL
```

The next example uses DELETED() in an INDEX statement to place all records marked as deleted at the beginning of the ordered table:

```
USE Clients EXCLUSIVE
DELETE FOR State_Prov = "CA"
INDEX ON IIF(DELETED(), "AA"+State_Prov, "Z" ;
+State_Prov)TAG DelState
BROWSE FIELDS Company, City, State_Prov
* California records would appear at the top; other
* states alphabetically below.
RECALL ALL
CLOSE ALL
```

See Also

DELETE, PACK, RECALL, SET DELETED

DESCENDING()

Table organization

Indicates if a specified index was created with the DESCENDING keyword.

Syntax

DESCENDING([<.mdx filename expC>.] <index position expN> [, <alias>])

<.mdx filename expC> Specifies a multiple index file that contains the index tag you want to check.

<index position expN> Selects an index tag by its position in an .MDX file or the position of an index tag in the list of open indexes in the current or specified work area.

DIFFERENCE()

<alias> A work area number (1 through 225), letter (A through J), or alias name. The work area letter or alias name must be enclosed in quotes.

Description

The DESCENDING() function returns .T. if the index tag specified by the *<index position expN>* parameter was created with the DESCENDING keyword; otherwise, it returns .F.. The optional alias parameter specifies the work area in which you want to check an index tag. If you do not specify a work area, the current work area is assumed.

If you do not specify an .MDX file name, the DESCENDING() function checks for the index tag in all open .MDX files. If a production .MDX file and other .MDX files are open, the DESCENDING() function checks for the DESCENDING keyword in the production .MDX file only.

If you do not specify an index tag, DESCENDING() determines if the master index tag was created with the DESCENDING keyword. DESCENDING() returns .F. if the index is an .NDX file or there is no master index.

If the specified .MDX file or index tag does not exist, the DESCENDING() function returns an error message.

Example

The following example uses DESCENDING() to determine if the key for a specified index was created using a descending index:

```
USE Company EXCLUSIVE
INDEX ON Company TAG Company
SET ORDER TO TAG Company
? TAG(), "Descending = ", DESCENDING()

INDEX ON Company TAG DescCo DESCENDING
? TAG(), "Descending = ", DESCENDING()
```

Portability

Not supported in dBASE III PLUS.

See Also

FOR(), INDEX, KEY(), MDX(), ORDER(), TAGCOUNT(), TAGNO(), UNIQUE()

DIFFERENCE()

String data

Returns a number that represents the phonetic difference between two strings.

Syntax

DIFFERENCE(*<expC1>* | *<memo field 1>*, *<expC2>* | *<memo field 2>*)

<expC1> | **<memo field 1>** The first character expression or memo field to evaluate the SOUNDEX() of and compare to the second value.

<expC2> | <memo field 2> The second character expression or memo field to evaluate the SOUNDEX() of and compare to the first value.

Description

SOUNDEX() returns a four-character code that represents the phonetic value of a character expression or memo field. DIFFERENCE() compares the SOUNDEX() codes of two character expressions or memo fields, and returns an integer from 0 to 4 that expresses the difference between the codes.

A returned value of 0 indicates the greatest difference in SOUNDEX() codes—the two expressions have no SOUNDEX() characters in common. A returned value of 4 indicates the least difference—the two expressions have all four SOUNDEX() characters in common. However, using DIFFERENCE() on short strings can produce unexpected results, as shown in the following example.

```
? SOUNDEX("Mom")      && returns M500
? SOUNDEX("Dad")      && returns D300
? DIFFERENCE("Mom","Dad") && returns 2
```

To compare the character-by-character similarity between two strings rather than the phonetic similarity, use LIKE().

Example

The following example uses DIFFERENCE() to determine how closely two text strings might match phonetically:

```
? SOUNDEX("Apples")      && Returns A142
? SOUNDEX("Lapels")      && Returns L142
? DIFFERENCE("Apples","Lapels") && Returns 3
? DIFFERENCE("Katherine","Kathryn") && Returns 4
? DIFFERENCE("Daniel","Damien") && Returns 3
? DIFFERENCE("Money","Monet") && Returns 3
? DIFFERENCE("dBASE","C") && Returns 1
? DIFFERENCE("Motor Oil","Cookies") && Returns 1
```

Portability

Not supported in dBASE III PLUS. The memo field arguments aren't supported in dBASE IV.

See Also

LIKE(), SOUNDEX()

DIR/DIRECTORY

Disk and file utilities

Performs a directory listing.

Syntax

DIR (or DIRECTORY)

[[ON]<drive>:]

[[LIKE] [<path>] [<filename> | <filename skeleton>]]

[ON] <drive>: Specifies the drive from which DIR performs its directory listing. A colon (:) is optional after ON <drive>; however, if you use <drive> without ON, you must follow it with a colon.

ON is optional and has no effect; it is included for backward compatibility with dBASE IV.

[LIKE] [<path>] <filename> | <filename skeleton> Specifies a path and/or file specification to be used by DIR. Use the <path> option to perform a directory listing in a directory other than the directory that is current on the default drive or on <drive>. Use the <filename> option to list a single file. Use the <filename skeleton> option to list multiple files.

LIKE is optional and has no effect; it is included for backward compatibility with dBASE IV.

Description

DIR (or DIRECTORY) is a utility command that lets you perform a directory listing. The information provided on each file includes its name, its size in bytes, and the date of its last update. DIR also shows the total number of bytes used by the listed files, the number of bytes left on disk, and the total disk space.

DIR with no arguments displays information only on files with .DBF extensions in the current directory; in addition to the information normally displayed, DIR displays the number of records in each database.

You can use the wildcard characters * and ? to identify multiple files with similar names. DIR with no arguments, DIR *.DBF, and DIR * are all equivalent commands. DIR N* displays all .DBF files beginning with N. DIR N???? displays all .DBF files that have five-letter names beginning with N. DIR N*.* displays all files beginning with N, regardless of extension. DIR N*. displays all files beginning with N that have no file-name extension.

If you want to specify a drive and path or file name information without using ON or LIKE, you must follow <drive> with a colon. DIR ON C LIKE *.BAT and DIR C:*.BAT are equivalent commands.

A path must end with a backslash (\), as shown in the following example:

```
* the following returns the .dbf file names
DIR C:\VISUALDB\SAMPLES\
* the following returns "0 bytes in 0 files"
DIR C:\VISUALDB\SAMPLES
```

If you have not used ON KEY or SET FUNCTION to reassign the *F4* key, pressing *F4* is a quick way to execute DIR.

The output presented by DIR is affected by the settings made by SET DATABASE and SET DBTYPE.

Example

The following examples use DIR:

```
DIR                                && Shows all tables in current subdirectory
DIR ON B:                          && Shows all tables on B drive
```

```

DIR A:                                && Shows all tables on A drive
DIR *.prg                            && Shows all program files
DIR LIKE *.* ON C:\DBASE
* Shows all files on C:\DBASE

```

Portability

The ON and LIKE options are not supported in dBASE III PLUS.

D**See Also**

DISPLAY FILES, FILE(), LIST FILES, FSIZE(), ON KEY, SET DATABASE, SET DEFAULT, SET DBTYPE TO, SET FUNCTION

DISKSPACE()

Disk and file utilities

Returns the number of bytes available on the default or specified drive's disk.

Syntax

DISKSPACE([<drive expN>])

<drive expN> A drive number from 1 to 26. For example, the numbers 1 and 2 correspond to drives A and B, respectively.

Without <drive expN> or if <drive expN> is 0, DISKSPACE() returns the number of bytes available on the default drive.

If <drive expN> is less than 0 or greater than 26, DISKSPACE() returns the number of bytes available on the drive from which you load dBASE.

Description

Use DISKSPACE() to determine how much space is left on a disk.

DISKSPACE() can be used in conjunction 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

The following examples use DISKSPACE():

```

? DISKSPACE()      && current drive
? DISKSPACE(2)     && drive B
? DISKSPACE(-1)    && dBASE

```

Portability

The optional argument <drive expN> is not supported in dBASE IV or dBASE III PLUS.

See Also

HOME(), RECCOUNT(), RECSIZE(), SET DEFAULT

Displays records from the current table in the result pane of the Command window.

Syntax

```

DISPLAY
[<scope>]
[FOR <condition 1>]
[WHILE <condition 2>]
[[FIELDS] <exp list>]
[OFF]
[TO FILE <filename> | ? | <filename skeleton>]
[TO PRINTER]

```

<scope> The number of records to DISPLAY. RECORD <*n*> identifies a single record by its record number. NEXT <*n*> identifies *n* records, beginning with the current record. ALL specifies all records. REST specifies all records from the current record to the end of the file.

FOR <condition 1>

WHILE <condition 2> Determines which records appear with DISPLAY. FOR restricts DISPLAY to records that meet <condition 1>, starting at the first record of the table or scope and continuing until the end of the table or scope. WHILE starts processing with the current record and continues with each subsequent record as long as <condition 2> is true.

FIELDS <exp list> Field names or expressions whose contents (values) you want to display; the names of the fields in the list are separated by commas. If you don't list any, Visual dBASE displays the values of all fields for the current record or the records you specify. The FIELDS keyword is included for readability only; it has no affect on the operation of the command.

OFF Suppresses display of the record number. Record numbers are not displayed for Paradox or SQL tables.

[TO FILE <filename> | ? | <filename skeleton>] Sends output to a file or to a printer. TO FILE directs output to the text file <filename>, as well as to the Command window (unless SET CONSOLE is OFF). By default, Visual dBASE assigns a .TXT extension to <filename> and saves the file in the current directory. The ? and <filename skeleton> options display a dialog box, in which you specify the name of the target file and the directory to save it in. TO PRINTER directs output to the printer, as well as to the results pane of the Command window.

[TO PRINTER] Directs output to the printer, as well as to the results pane of the Command window.

Description

Use DISPLAY to view one or more records of the current table in the results pane of the Command window. Issuing DISPLAY without any arguments displays only the current record whereas LIST displays all records. If SET HEADINGS is OFF, Visual dBASE doesn't display field names when you issue DISPLAY.

DISPLAY pauses when the results pane is full and displays a dialog box prompting you to display additional information. Scroll the output up or down by doing any of the following:

- Press *Enter* or *PgDn* to scroll down to the next window.
- Click near the bottom of the vertical scroll bar to scroll down to the next window.
- Press *PgUp* to scroll up to the next window.
- Click near the top of the vertical scroll bar to scroll up to the next window.
- Press the Up or Down arrows to scroll up or down, respectively, one line.
- Click the vertical scroll bar up or down arrows to scroll up or down, respectively, one line.
- Press *Spacebar* to scroll up one line.
- Type *<expN>* and press *Enter* to scroll up the specified number of lines.

DISPLAY is similar to LIST, except that LIST doesn't pause at the end of the first window of information but rather lists the information continuously, halting at the last window.

If the information output to the results pane is more than the dBASE buffer can contain, you might not be able to scroll back up to information you've scrolled down through. Use the TO FILE option to send the information to a file. Use the TO PRINTER option to send the information to a printer.

Example

The following example uses DISPLAY to display selected data from the Clients table:

```
USE Clients
DISPLAY
* display all fields in the current record

DISPLAY ALL
* display all fields in all records. On completion
* record counter is positioned at end of table

GOTO 2      && go to 2nd record in the table
DISPLAY REST FIELDS Company, Contact
* display company and contact fields in rest of table

SET FIELDS TO Company, Contact, Phone
DISPLAY all
* combine the set fields and display commands
* to display company, contact and phone fields
* in all records
```

See Also

DISPLAY STRUCTURE, LIST, SET HEADINGS, SET FIELDS

DISPLAY COVERAGE

Error handling and debugging

Displays the contents of a coverage file (.COV) in the results pane of the Command window.

Syntax

DISPLAY COVERAGE

<.COV filename> | ? | <filename skeleton 1>

[ALL]

[SUMMARY]

[TO FILE <filename> | ? | <filename skeleton>] | [TO PRINTER]

<.COV filename> | ? | <filename skeleton> The coverage file that dBASE creates or modifies when SET COVERAGE is ON, and you call a program, or call a procedure or UDF in a procedure file. The ? and <filename skeleton> options display a dialog box from which you can select a coverage file. If you specify a file without including its path, dBASE looks for the file in the current directory, then in the path you specify with SET PATH. If you specify a file without including its extension, dBASE assumes .COV.

ALL Displays in the results pane of the Command window the following information:

- The contents of the coverage file for <filename 1>'s program and of the coverage files for each executed program or procedure file called by <filename 1>'s program
- The total number of logical blocks entered and exited in all the program files combined
- The percentage of logical blocks entered and exited in all the program files combined

SUMMARY If ALL isn't specified, displays the following information about <filename 1>'s program. If ALL is specified, displays the following information about <filename 1>'s program and about all programs called by <filename 1>'s program:

- The logical blocks *not* entered and exited
- The total number of logical blocks entered and exited
- The percentage of logical blocks entered and exited

TO FILE <filename 2> | ? | <filename skeleton> Directs output to <filename 2> and to the results pane of the Command window. By default, dBASE assigns a .TXT extension to <filename 2> and saves the file in the current directory. The ? and <filename skeleton> options display a dialog box in which you specify the name of the target file and the directory to save it in.

TO PRINTER Directs output to the printer and to the results pane of the Command window.

Description

A coverage file contains the results of the coverage analysis of a program, procedure, or UDF. You cause dBASE to analyze a program, procedure, or UDF and store the results of the analysis in a coverage file with the SET COVERAGE ON command. You can issue DISPLAY COVERAGE in a program file or in the Command window.

DISPLAY COVERAGE displays the contents of *<filename 1>* and the results of percentage calculations using *<filename 1>*'s contents. The output appears in the results pane of the Command window, starting at the beginning of the main program's coverage file. DISPLAY COVERAGE displays the first window of output and pauses when the window is full. See the description of DISPLAY for information about how to navigate in the window.

The amount of memory available determines the size of the results pane of the Command window buffer. If the information output is more than the buffer can contain, you might not be able to scroll back up to information that has already been displayed. In that case, use either the TO FILE or TO PRINTER option to send all information to a file or printer.

DISPLAY COVERAGE is the same as LIST COVERAGE, except that LIST COVERAGE doesn't pause with the first window of information but rather continuously lists the information, halting at the last window. As with DISPLAY COVERAGE output, you can scroll LIST COVERAGE output.

Example

See SET COVERAGE for an example of using DISPLAY COVERAGE.

Portability

Not supported in dBASE IV or dBASE III PLUS.

See Also

#pragma, DISPLAY, LIST, SET COVERAGE

DISPLAY FILES

Disk and file utilities

Displays information about files on disk in the results pane of the Command window.

Syntax

DISPLAY FILES

[[LIKE] *<filename 1>* | *<filename skeleton 1>*]

[ON *<drive>*]

[TO FILE *<filename 2>* | ? | *<filename skeleton>*] | [TO PRINTER]

[LIKE] *<filename 1>* | *<filename skeleton>* The file you want to display information on. DISPLAY FILES *<filename skeleton>* displays the names of all files matching a template that uses the wildcard characters ? and *. If you specify a file, and the file is not in the default directory or in the path you specify with SET PATH, then *<filename>* must include the drive and path. LIKE has no effect on the operation of the command; include it for readability only.

ON *<drive>* The disk drive containing the files you want to display information on. By default, dBASE looks for the file or files you specify in the current drive and directory.

TO FILE *<filename 2>* | ? | *<filename skeleton>* Directs output to the text file *<filename 2>* and to the results pane of the Command window. By default, dBASE assigns a .TXT extension

to *<filename 2>* and saves the file in the current directory. The ? and *<filename skeleton>* options display a dialog box in which you specify the name of the target file and the directory to save it in.

TO PRINTER Directs output to the printer and to the results pane of the Command window.

Description

DISPLAY FILES is a utility command that lets you perform a directory listing. The information provided on each file includes its name, its size in bytes, and the date of its last update. DISPLAY FILES also shows the total number of bytes used by the listed files, the number of bytes left on disk, and the total disk space.

DISPLAY FILES with no options lists only table files (.DBF) in the current default directory. The number of records is displayed for each table, along with its size in bytes, and the date of its last update.

You can use the wildcard characters * and ? to identify multiple files with similar names. For example, although DISPLAY FILES * is equivalent to DISPLAY FILES with no argument (only .DBF files are displayed), DISPLAY FILES N* displays all .DBF files beginning with N. DISPLAY FILES N*.* displays all files beginning with N, regardless of extension. DISPLAY FILES N????.DBF displays all .DBF files that have five-letter names beginning with N.

DISPLAY FILES pauses at the start of the output when the window is full. See the description of DISPLAY for information about how to navigate in the window.

If the information output is more than dBASE's buffer can contain, you might not be able to scroll back up to information you've scrolled down through. Use either the TO FILE or TO PRINTER option to send all information to a file or printer.

DISPLAY FILES is the same as LIST FILES, except that LIST FILES doesn't pause with the first window of information but rather continuously lists the information, halting at the last window. As with DISPLAY FILES output, you can scroll LIST FILES output.

The SET DATABASE and SET DBTYPE settings affect DISPLAY FILES and LIST FILES output.

Example

The following examples use DISPLAY FILES:

```
DISPLAY FILES
* Shows all tables in current subdirectory
DISPLAY FILES ON B: && Shows all tables on B drive
DISPLAY FILES A: && Shows all tables on A drive
DISPLAY FILES *.prg && Shows all program files
DISPLAY FILES LIKE *.* ON C:\DBASE
* Shows all files on C:\DBASE
```

Portability

Not supported in dBASE III PLUS. The ON *<drive>* and TO FILE ? options are not supported in dBASE IV.

In dBASE IV, if you use TO FILE <filename 2> and do not include an extension, dBASE IV adds a .PRT extension. *Visual* dBASE adds a .TXT extension instead.

See Also

DIR, DISPLAY, FILE(), LIST, SET DATABASE, SET DBTYPE, SET DEFAULT

D

DISPLAY MEMORY

Environment

Displays information about memory variables in the results pane of the Command window.

Syntax

DISPLAY MEMORY

[TO FILE <filename> | ? | <filename skeleton>] | [TO PRINTER]

TO FILE <filename> | ? | <filename skeleton> Directs output to the text file <filename>, also called the target file, and to the results pane of the Command window. By default, dBASE assigns a .TXT extension to <filename> and saves the file in the current directory. The ? and <filename skeleton> options display a dialog box in which you specify the name of the target file and the directory to save it in.

TO PRINTER Directs output to the printer and to the results pane of the Command window.

Description

Use DISPLAY MEMORY to display the contents and size of a memory variable list. If you haven't used ON KEY or SET FUNCTION to reassign the *F7* key, pressing *F7* is a quick way to execute DISPLAY MEMORY.

DISPLAY MEMORY displays information about both user-defined and system memory variables. The following information on user-defined memory variables is displayed.

- Name
- Scope (public, private, local, static or hidden)
- Data type
- Value
- Number of active memory variables
- Number of memory variables still available for use
- Number of bytes of memory used by character variables
- Number of bytes of memory still available for user character variables
- Name of the program that initialized private memory variables

The following information on system memory variables is displayed.

- Name
- Scope (public, private, or hidden)
- Data type
- Current value

DISPLAY MEMORY displays the first window of output and pauses when the window is full. See the description of DISPLAY for information about how to navigate in the window.

The amount of memory available determines the size of the results pane of the Command window buffer. If the information output is more than the buffer can contain, you might not be able to scroll back up to information that has already been displayed. In that case, use either the TO FILE or TO PRINTER option to send all information to a file or printer.

If SET SAFETY is ON and a file exists with the same name as the target file, dBASE displays a dialog box asking if you want to overwrite the file. If SET SAFETY is OFF, any existing file with the same name is overwritten without warning.

DISPLAY MEMORY is the same as LIST MEMORY, except that LIST MEMORY displays the information without pauses, halting at the last window. As with DISPLAY MEMORY output, you can scroll LIST MEMORY output.

Portability

Only a TO PRINT option is supported in dBASE III PLUS. The ? and <filename skeleton> options are not supported in dBASE IV. If you don't specify a file-name extension with the TO FILE <filename> option, dBASE IV adds .PRT instead of .TXT.

See Also

LIST, LOCAL, PRIVATE, PUBLIC, SET FUNCTION, SET SAFETY, ON KEY, STATIC, STORE

DISPLAY STATUS

Environment

Displays information about the current dBASE environment in the results pane of the Command window.

Syntax

DISPLAY STATUS
[TO FILE <filename> | ? | <filename skeleton>] | [TO PRINTER]

TO FILE <filename> | ? | <filename skeleton> Directs output to the text file <filename>, also called the target file, and to the results pane of the Command window. By default, dBASE assigns a .TXT extension to <filename> and saves the file in the current directory. The ? and <filename skeleton> options display a dialog box in which you specify the name of the target file and the directory to save it in.

TO PRINTER Directs output to the printer and to the results pane of the Command window.

Description

Use DISPLAY STATUS to identify open tables and index files and to check the status of the SET commands. DISPLAY STATUS shows information related to the current session only.

If you haven't used ON KEY, SET, or SET FUNCTION to reassign the *F6* key, pressing *F6* is a quick way to execute DISPLAY STATUS.

DISPLAY STATUS displays the following information:

- Name and alias of open tables in each work area
- Names of all open memo files in each work area
- Name of any open format file in each work area
- Names of all open indexes and their index key expressions in each work area
- SET ORDER setting in each work area
- Master index, if any, in each work area
- Database relations in each work area
- Filter conditions in each work area
- SET PATH file search path
- SET DEFAULT drive setting
- Current work area
- SET PRINTER or SET DEVICE setting
- DBTYPE setting
- Numeric settings for SET MARGIN, SET DECIMALS, SET MEMOWIDTH, SET TYPEAHEAD, SET ODOMETER, SET REFRESH, and SET REPROCESS
- ON KEY, ON ESCAPE, and ON ERROR settings
- SET ON/OFF command settings
- Programmable function key and SET FUNCTION settings

D

DISPLAY STATUS displays the first window of output and pauses when the window is full. See the description of DISPLAY for information about how to navigate in the window.

The amount of memory available determines the size of the Results pane of the Command window buffer. If the information output is more than the buffer can contain, you might not be able to scroll back up to information that has already been displayed. In that case, use either the TO FILE or TO PRINTER option to send all information to a file or printer.

If SET SAFETY is ON and a file exists with the same name as the target file, dBASE displays a dialog box asking if you want to overwrite the file. If SET SAFETY is OFF, any existing file with the same name is overwritten without warning.

DISPLAY STATUS is the same as LIST STATUS, except that LIST STATUS displays the information without pauses, halting at the last window. As with DISPLAY STATUS output, you can scroll LIST STATUS output.

Portability

Only a TO PRINT option is supported in dBASE III PLUS. The ? and <filename skeleton> options are not supported in dBASE IV. If you don't specify a file-name extension with the TO FILE <filename> option, dBASE IV adds .PRT instead of .TXT.

See Also

LIST, SET(), SETTO(), SET SAFETY

DISPLAY STRUCTURE

Table basics

Displays the field definitions of the specified table.

Syntax

DISPLAY STRUCTURE

[IN <alias>]

[TO FILE <filename> | ? <filename skeleton>] | [TO PRINTER]

IN <alias> Identifies the work area of the open table whose structure you want to display rather than that of the current table. You can enter a work area number (1 through 225), letter (A through J), or alias name. The work area letter or alias name must be enclosed in quotes.

[TO FILE <filename> | ? | <filename skeleton>] | [TO PRINTER] Sends output to a file or to a printer. TO FILE directs output to the text file <filename>, as well as to the results pane of the Command window. By default, *Visual* dBASE assigns a .TXT extension to <filename> and saves the file in the current directory. The ? and <filename skeleton> options display a dialog box, in which you specify the name of the target file and the directory to save it in. TO PRINTER directs output to the printer, as well as to the results pane of the Command window.

Description

Use DISPLAY STRUCTURE to view the structure of the current or a specified table in the results pane of the Command window. DISPLAY STRUCTURE displays the following information about the current or specified table:

- Name of the table
- Type of table (Paradox, dBASE, or SQL)
- Number of records
- Date of last update
- Fields
- Field number
- Field name (if SET FIELDS is ON, the greater-than symbol (>) appears next to each field specified with the SET FIELDS TO command)
- Type
- Length
- Number of bytes per record (the sum of field lengths plus one additional byte reserved for storing the asterisk that marks a record for deletion)

Multiply the total number of bytes per record by the number of records in the table to estimate the size of a dBASE table (excluding the size of the table file header).

DISPLAY STRUCTURE pauses when the results pane of the Command window is full and displays a dialog box prompting you to display additional information. See the description of DISPLAY for information about how to navigate in the Command window.

If the information output to the results pane of the Command window is more than the *Visual* dBASE buffer can hold, you might not be able to scroll back up to information

you've scrolled down through. Use the TO FILE or TO PRINTER options to save all information to a file or as printer output. You can send output to either a file or printer, but not both at the same time.

DISPLAY STRUCTURE is similar to LIST STRUCTURE, except that LIST STRUCTURE doesn't pause when the Command window fills, but rather lists the information continuously.

D

Neither DISPLAY STRUCTURE nor LIST STRUCTURE permit modification of an existing table structure. To alter the structure, use MODIFY STRUCTURE.

Example

The following example uses DISPLAY STRUCTURE and LIST STRUCTURE to display the structure of a table:

```
USE Clients
LIST STRUCTURE TO FILE Clients.TXT
```

The following text is saved in a text file named CLIENTS.TXT by the LIST STRUCTURE command:

```
Structure for table   : C:\VISUALDB\SAMPLES\CLIENTS.DBF
Table type           : DBASE
Number of records    : 100
Last update          : 05/01/94
```

Field	Field Name	Type	Length	Dec	Index
1	CLIENT_ID	CHARACTER	5		N
2	COMPANY	CHARACTER	35		N
3	CONTACT	CHARACTER	20		N
4	ADDRESS	CHARACTER	30		N
5	CITY	CHARACTER	15		N
6	STATE	CHARACTER	2		N
7	ZIP	CHARACTER	5		N
8	AREACODE	CHARACTER	3		N
9	PHONE	CHARACTER	8		N
10	EXTENSION	CHARACTER	5		N
11	STARTBAL	NUMERIC	8	2	N
12	BALDATE	DATE	8		N
13	CUISINE	CHARACTER	15		N
14	TYPE	NUMERIC	1		N
15	NOTES	MEMO	10		N
** Total **			171		

See Also

DISPLAY, LIST, MODIFY STRUCTURE, SET FIELDS

DMY()

Returns a specified date expression as a character string in DD MONTH YY or DD MONTH YYYY format.

Syntax

DMY(<expD>)

<expD> The date expression to return as a character string in DD MONTH YY or DD MONTH YYYY format.

Description

DMY() returns a date in DD MONTH YY or DD MONTH YYYY format, where DD is the day number, MONTH is the full month name, and YY is the year number. If SET CENTURY is OFF (the default), DMY() returns the year as 2 digits. If SET CENTURY is ON, DMY() returns the year as 4 digits.

Enter <expD> in the current date format as determined by SET DATE, DBASEWIN.INI, or the International option of the Windows Control Panel (in that order). That is, SET DATE settings override those in DBASEWIN.INI, and settings in DBASEWIN.INI override those in the Windows Control Panel. Be sure <expD> matches the date format in use when your program runs.

If you pass an invalid date to DMY(), it converts the date to a valid one and returns that date in DD MONTH YY or DD MONTH YYYY format. If you pass an empty or non-date expression delimited with braces ({ }) to DMY(), it returns "0 Unknown 00" or "0 Unknown 0000". If you pass a non-date expression or an expression that isn't delimited with braces to DMY(), dBASE returns an error.

Example

The following example uses DMY() to display a date field value in the form day number, month spelled out and year as determined by SET CENTURY:

```
SET TALK OFF
USE Clients
Cnt=1
DO WHILE .NOT. EOF()
CLEAR
? CENTER("Client Database Report",80)
? CENTER("Run on " + CDOW( DATE() ) + ", " + DMY( DATE() ),80,"-")
?
? "Company" AT 2, "Phone" AT 40, "First Contact" AT 55
DO WHILE Cnt <= 10 .AND. .NOT. EOF()
? Company AT 2, Phone AT 40, CDOW( Baldate ) + ", " + DMY( Baldate ) AT 55
SKIP
Cnt=Cnt+1
ENDDO
Cnt=1
WAIT
ENDDO
CLOSE ALL
```

Portability

Not supported in dBASE III PLUS.

See Also

CROW(), CMONTH(), DAY(), DOW(), MDY(), MONTH(), SET CENTURY, SET DATE, YEAR()

D

DO**Programs**

Runs a program, procedure, or user-defined function (UDF).

Syntax

```
DO <filename> | ? | <filename skeleton> |  
<procedure name> |  
<UDF name>  
[WITH <parameter list>]
```

<filename> | ? | <filename skeleton> The program file to execute. The ? and <filename skeleton> options display a dialog box from which you can select a file. If you specify a file without including its path, dBASE looks for the file in the current directory, then in the *search path* in *search order*. See "Search path and order" later in this section for more information.

If you specify a file without including its extension, dBASE assumes a .PRO extension (a compiled object file). If dBASE can't find a .PRO file, it looks for a .PRG file (a source file), which, if found, it compiles. By default, dBASE creates the .PRO in the same directory as the .PRG, which might not be the current directory.

<procedure name> | <UDF name> The procedure or UDF in an open program file to execute. The procedure or UDF must be in the program file containing the DO command that calls it, or in a separate open file on the search path. The search path is described later in this section.

WITH <parameter list> Specifies memory variable values, field values, or any other valid expressions to pass as parameters to a program, procedure, or UDF. See the description of PROCEDURE for information on parameter passing.

Description

Use DO to run programs from the Command window or to run procedures or other programs from a program. If you enter DO in the Command window, control returns to the Command window when the program, procedure, or UDF ends. If you use DO in a program to execute another program, procedure, or UDF, control returns to the program line following the DO statement when the program, procedure, or UDF ends.

The limit on the number of nested DOs calls depends on the amount of available memory and the number of files you have open (a DO command might open a file). Avoid using DO recursively because you can quickly exceed the limit of nested DOs.

When dBASE encounters a DO statement in a program file, it looks in that file for a procedure or UDF of the specified name. If the current program file contains a

procedure and a UDF with the same name, dBASE executes the first one declared. If dBASE doesn't find a PROCEDURE or FUNCTION definition of the specified name in the same program file, it looks for a program, procedure, or UDF of the specified name on the *search path* in *search order*.

Search path and order

If the name you specify with DO doesn't include a path or a file-name extension, it can be a program, procedure, UDF, or file name. To resolve the ambiguity, dBASE searches for the name in specific places (the search path) in a specific order (the search order) and runs the first program or subroutine of the specified name that it finds. The search path and order dBASE uses is as follows:

- 1 The executing program's object file (.PRO)
- 2 Other open object files (.PRO) in the call chain, in most recently opened order
- 3 The file specified by SYSPROC = <filename> in DBASEWIN.INI
- 4 Any files opened with SET PROCEDURE, SET PROCEDURE...ADDITIVE, or SET LIBRARY statements, in the order in which they were opened
- 5 The object file (.PRO) with the specified name in the search path
- 6 The program file (.PRG) with the specified name in the search path, which dBASE automatically compiles

Example

The following example uses DO to run procedures:

```
DO Client_Rpt
DO SalesReport WITH "CA"
DO AddRecord WITH "Smith", "John", 25

PROC Client_Rpt
* ...
RETURN

PROC SalesReport
Parameters State
* ...
RETURN

PROC AddRecord
PARAMETERS Lastname, Firstname, Age
* ...
RETURN
```

Portability

Calling a UDF with DO isn't supported in dBASE IV or dBASE III PLUS. The search path and search order of dBASE IV and dBASE III PLUS differ from that of *Visual* dBASE. Both dBASE IV and dBASE III PLUS create compiled object files with .DBO extensions, and place them in the current directory, by default.

See Also

!, CLEAR PROGRAM, COMPILE, DOS, PROCEDURE, RETURN, RUN, RUN(), SET DEVELOPMENT, SET ESCAPE, SET LIBRARY, SET PROCEDURE

DO CASE

Programs

D

Conditionally processes statements by evaluating one or more conditions and executing the statements following the first condition that evaluates to true.

Syntax

```
DO CASE
CASE <condition expl 1>
    <statements>
[CASE <condition expl 2>
    <statements>...]
[OTHERWISE
    <statements>]
ENDCASE
```

CASE <condition expl> If the condition is true, executes the set of commands between CASE and the next CASE, OTHERWISE, or ENDCASE command, and then transfers control to the line following ENDCASE. If the condition is false, control transfers to the next CASE, OTHERWISE, or ENDCASE command.

<statements> One or more program lines consisting of any combination of commands, functions, and user-defined functions (UDFs).

OTHERWISE Executes a set of statements if all the CASE statements evaluate to false.

ENDCASE A required command that marks the end of the DO CASE structure.

Description

DO CASE is similar to IF...ELSE...ENDIF. As with IF conditions, dBASE evaluates DO CASE conditions in the order they're listed in the structure. However, DO CASE acts on only the first true condition in the structure, even if several apply. In situations where you want only the first true instance to be processed, use DO CASE instead of a series of IF commands.

Also, use DO CASE when you want to program a number of exceptions to a condition. The CASE <condition> statements can represent the exceptions, and the OTHERWISE statement the remaining situation.

Starting with the first CASE condition, dBASE does the following.

- Evaluates each CASE condition until it encounters one that's true
- Executes the statements between the first true CASE statement and the next CASE, OTHERWISE, or ENDCASE
- Exits the DO CASE structure without evaluating subsequent CASE conditions
- Moves program control to the first line after the ENDCASE command

If none of the conditions are true, dBASE executes the statements under OTHERWISE if it's included. If no OTHERWISE statement exists, dBASE exits the structure without executing any statements and transfers program control to the first line after the ENDCASE command.

If you include command pairs such as IF...ENDIF and DO WHILE...ENDDO, you must nest them within separate DO CASE statements. You can also nest DO CASE structures.

Example

Compare this example with the examples of the IF command. The following CASE construct determines the magnitude of a variable and displays an appropriate message:

```
nM_value = 225
DO CASE
  CASE nM_value > 1000
    ? "Value is over 1000."
  CASE nM_value > 100
    ? "Value is over 100."
  CASE nM_value > 10
    ? "Value is over 10."
  CASE nM_value > 1
    ? "Value is over 1."
  OTHERWISE
    ? "The value is 1 or less."
ENDCASE
```

See Also

DO WHILE, DO...UNTIL, FOR...NEXT, IF, IIF(), SCAN

DO WHILE

Programs

Executes the statements between DO WHILE and ENDDO as long as a specified condition is true or until dBASE encounters an EXIT command.

Syntax

```
DO WHILE <condition expl>
<statements>
[LOOP]
[EXIT]
ENDDO
```

<condition expl> A logical expression that determines if dBASE executes the statements following the DO WHILE statement. If the expression evaluates to .F., dBASE skips the statements following DO WHILE and executes the command line following ENDDO. If the expression evaluates to .T., dBASE executes the commands in the DO WHILE loop.

<statements> Program lines consisting of any combination of commands, functions, user-defined functions (UDFs), and LOOP and EXIT options.

LOOP Returns program control to the top of the DO WHILE loop without executing the statements that follow LOOP and precede ENDDO.

EXIT Transfers program control out of the DO WHILE loop to the command following ENDDO without executing the statements that follow EXIT and precede ENDDO.

ENDDO A required command that marks the end of the DO WHILE loop.

Description

Use DO WHILE...ENDDO to execute specified statements for as many times as a specified condition is true. When dBASE encounters a DO WHILE statement, it determines whether *<condition expL>* is true or false. If *<condition expL>* is true, dBASE executes the statements in the DO WHILE loop one by one until it encounters ENDDO, LOOP, or EXIT. If *<condition expL>* is false, dBASE skips the loop and executes the command line following ENDDO.

ENDDO and LOOP return program control to the DO WHILE statement and cause dBASE to evaluate *<condition expL>* again. EXIT transfers program control out of the DO WHILE loop to the command following ENDDO.

You can nest loops and other structures, including other DO WHILE loops, in a DO WHILE loop. You can nest up to 125 DO WHILE loops in a procedure or function; the maximum depends on the number of other loops, such as DO UNTIL, that are also running..

DO WHILE...ENDDO and DO...UNTIL are opposite constructs. DO WHILE...ENDDO executes commands while a condition is true (until the condition is false), while DO...UNTIL executes commands while a condition is false (until the condition is true). Because DO...UNTIL executes commands before evaluating the condition, the commands between DO and UNTIL execute at least once, even when the condition is true. Because DO WHILE...ENDDO evaluates the condition before executing commands, the commands between DO WHILE and ENDDO don't execute at all if the condition is initially false.

Example

The following example uses DO WHILE to step through a table one record at a time and bring up the edit window if a record is missing a phone number. This process repeats until the EOF() marker is encountered or the user answers "N" to the continue prompt:

```
SET EXCAPE OFF
USE Clients
CLEAR
DO WHILE .NOT. EOF()
  IF ISBLANK(Phone)
    mRec = Recno()
    EDIT NOAPPEND NODELETE RECORD MREC
    SKIP
    ACCEPT "Continue?(Y/N)" TO mCont
    IF UPPER(mCont) = "N"
      EXIT
    ENDIF
    CLEAR
  ELSE
    SKIP
  ENDIF
ENDIF
```

ENDDO
RETURN

See Also

DO CASE, DO...UNTIL, FOR...NEXT, IF, SCAN

DO...UNTIL

Programs

Executes the statements between DO and UNTIL as long as a specified condition is false or until dBASE encounters an EXIT command.

Syntax

```
DO
<statements>
[LOOP]
[EXIT]
UNTIL <condition expl>
```

<statements> Program lines consisting of any combination of commands, functions, user-defined functions (UDFs), and LOOP and EXIT options.

LOOP Passes program control to the end of the DO...UNTIL loop and evaluates *<condition expl>* without executing the commands that follow LOOP and precede UNTIL.

EXIT Transfers program control out of the DO...UNTIL loop to the command following UNTIL *<condition expl>* without reevaluating *<condition expl>* or executing the commands that follow EXIT and precede UNTIL.

UNTIL <condition expl> A required command that marks the end of the DO...UNTIL loop. The *<condition expl>* argument is a logical expression that determines if dBASE executes the statements in the DO...UNTIL loop. If the expression evaluates to .F., program control returns to the DO command and dBASE executes the commands in the DO...UNTIL loop. If the expression evaluates to .T., program control transfers out of the loop to the line following UNTIL.

Description

Use DO...UNTIL to execute specified statements for as many times as a specified condition is false. When dBASE first encounters a DO...UNTIL loop, it executes the statements between DO and UNTIL until it encounters LOOP, EXIT, or UNTIL. If it encounters UNTIL, dBASE evaluates the condition to determine if it goes through the loop again. Because dBASE evaluates the condition after executing the statements, the statements in a DO...UNTIL loop always execute at least once, even when the condition is initially true.

You can nest loops and other structures, including other DO...UNTIL loops, in a DO...UNTIL loop. You can nest up to 116 DO...UNTIL loops in a procedure or function; the maximum depends on the number of other loops, such as DO WHILE, that are also running.

DO...UNTIL and DO WHILE...ENDDO are opposite constructs. DO...UNTIL executes commands while a condition is false (until the condition is true), while DO WHILE...ENDDO executes commands while a condition is true (until the condition is false). Unlike DO...UNTIL, DO WHILE...ENDDO evaluates the condition before executing commands; thus, the commands between DO WHILE and ENDDO don't execute at all if the condition is initially false.

Use DO...UNTIL when you want commands to execute at least once, as with data entry programs in which you require at least one entry and allow any number of further entries. This type of program often requires an explicit response to re-execute the commands in the DO...UNTIL loop. The program evaluates this response with UNTIL *<condition expL>*.

In other situations, the choice between DO...UNTIL and DO WHILE can also depend on which command saves a few program lines or makes a program more readable. You can often rephrase a negative condition for one into a positive condition for the other.

Example

The following example uses DO...UNTIL to show one screen full of information about flights. The DO ... UNTIL loop ends either at row 20 or when the end of file is reached:

```
SET TALK OFF
USE Flights EXCLUSIVE
CLEAR
? CENTER("Flight Information")
? "Flight", "Origin", "Dest", "Departure", "Arrival"
DO
    ? Flight_no, Origin, Dest, Departure, Arrival
SKIP
UNTIL row() >= 20 .OR. EOF()
CLOSE DATABASE
```

Portability

Not supported in dBASE IV or dBASE III PLUS.

See Also

DO WHILE, IF, FOR...NEXT

DOS

Disk and file utilities

Interrupts dBASE and displays the DOS prompt, allowing execution of DOS commands. dBASE resumes when EXIT is typed at the DOS prompt.

Syntax

DOS

Description

Use the DOS command to temporarily leave dBASE and perform commands in DOS. Enter commands in DOS as you normally do and use EXIT to return to dBASE after the desired commands have been executed.

When you issue commands that run in a DOS window, dBASE loads COMMAND.COM using DBASEWIN.PIF in the _dbwinhome directory. You can use the Windows PIF Editor to customize the settings in DBASEWIN.PIF. For more information about .PIF files, see your Windows documentation.

To execute single DOS commands without exiting dBASE, use ! or RUN. To execute Windows applications, use RUN().

Example

DOS takes you out to DOS:

```
DOS
```

Type EXIT from the DOS prompt to return to dBASE.

Portability

Not supported in dBASE IV or dBASE III PLUS.

See Also

_dbwinhome, RUN, RUN()

DOW()

Date and time data

Returns the day of the week corresponding to a specified date expression as a number from 1 to 7.

Syntax

DOW(<expD>)

<expD> The date expression, in the current date format, whose corresponding weekday number to return.

Description

DOW() returns the number of the day of the week on which a date falls. Sunday is equal to 1, Monday to 2, and so on through Saturday, which is equal to 7. To return the *name* of the day of the week instead of the number, use CDOW().

You must enter <expD> in the current date format as determined by SET DATE, DBASEWIN.INI, or the International option of the Windows Control Panel (in that order). That is, SET DATE settings override those in DBASEWIN.INI, and settings in DBASEWIN.INI override those in the Windows Control Panel. Be sure <expD> matches the date format in use when your program runs.

If you pass an invalid date to DOW(), dBASE converts the date to a valid one and returns the weekday number of that date. If you pass an empty or non-date expression

delimited with to DOW(), it returns 0. If you pass a non-date expression or an expression that isn't delimited with braces to DOW(), it returns an error.

Example

The following example determines how many records in the flight table have a Date field value that equates to Saturday (value of 7) or Sunday (value of 1):

```
CLEAR
USE Flights
COUNT FOR DOW(Date)=1 .OR. DOW(Date)=7 TO Weekend
? Weekend
RETURN
```

See Also

CDOW(), CMONTH(), SET CENTURY, SET DATE

D

DTC()

Expressions and type conversion

Returns a specified date expression as a character expression.

Syntax

DTC(<expD>)

<expD> The date expression, in the current date format, to return as a character expression.

Description

Use DTC() to convert a date expression to a character expression. Once you convert date data to character data, you can manipulate it as you would any string. For example, if you want to print the current date on a report, use DTC(DATE()). To convert a date expression to a character string suitable for indexing or sorting, use DTOS().

Specify <expD> in the current date format. dBASE uses the value set by SET DATE, DBASEWIN.INI, or the International option of the Windows Control Panel (in that order) to determine the value of <expD>. That is, SET DATE settings override those in DBASEWIN.INI, and settings in DBASEWIN.INI override those in the Windows Control Panel. Be sure <expD> matches the date format in use when your program runs. DTC() returns a character expression in the current date format.

If you pass an invalid date to DTC(), it converts the date to a valid one and returns that date as a character expression. If you pass an empty date ({ / / } or { }) to DTC(), it returns an empty string expression in the current date format. For example, if the SET DATE format is AMERICAN, DTC({ }) returns " / / ". If you pass a non-date expression or an expression that isn't delimited with braces ({ }) to DTC(), it returns an error.

Example

The following example uses DTC() to convert date values in the BalDate field to a character string so that they can be concatenated with a character field (Client_ID) in an

DTOR()

array element variable. Also note the conversion of a numeric value (StartBal) to character with STR(). The example program stores a three-field string for each record to array String, then outputs the array contents to the Command window results pane:

```
SET TALK OFF
USE Clients
DECLARE String[RECCOUNT()]
Cnt = 1
SCAN
    STRING[Cnt]=Client_ID+"*"+DTOC(BalDate)+"*"+ LTRIM(STR(StartBal,7,2))
    Cnt=Cnt+1
ENDSCAN
* The following FOR...NEXT loop verifies the contents of the array.
FOR i=1 TO RECCOUNT()
    ? STRING[i] AT 10
NEXT i
```

See Also

CTOD(), DATE(), DTOS(), SET DATE, SET CENTURY

DTOR()

Numeric data

Returns the radian value of an angle whose measurement is given in degrees.

Syntax

DTOR(<expN>)

<expN> A negative or positive integer or float that is the size of the angle in degrees.

Description

DTOR() converts the measurement of an angle from degrees to radians. DTOR() returns a float. To convert degrees to radians, dBASE

- Multiplies the number of degrees by pi
- Divides the result by 180
- Returns the quotient

A 180-degree angle is equivalent to pi radians.

Use DTOR() in the trigonometric functions SIN(), COS(), and TAN() because these functions require the angle value in radians. For example, to find the sine of a 45-degree angle, use SIN(DTOR(45)), which returns .71 if the default number of decimal places is 2.

Use SET DECIMALS to set the number of decimal places DTOR() displays.

Example

The following examples show some ways to use DTOR():

```
? DTOR(180)      && Returns 3.14 (pi)
? DTOR(0)        && Returns 0
```



```
? DTOR(360)    && Returns 6.28 (2*pi)
? DTOR(90)     && Returns 1.57 (pi/2)
? DTOR(-90)    && Returns -1.57
```

The following example converts 60 degrees 30 minutes 15 seconds to radians:

```
? DTOR(60.525) && Returns 1.06
```

D

Portability

Not supported in dBASE III PLUS.

See Also

ACOS(), ASIN(), ATAN(), ATN2(), COS(), PI(), RTOD(), SET DECIMALS, SIN(), TAN()

DTOS()

Expressions and type conversion

Returns a specified date expression as a character string in YYYYMMDD format.

Syntax

DTOS(<expD>)

<expD> The date expression, in the current format, to return as a character string in YYYYMMDD format.

Description

Use DTOS() to convert a date expression to a character string suitable for indexing or sorting. For example, you can use DTOS() when indexing on a date field in combination with another field of a different type. DTOS() always returns a character string in YYYYMMDD format, even if SET CENTURY is OFF.

You must enter <expD> in the current date format as determined by SET DATE, DBASEWIN.INI, or the International option of the Windows Control Panel (in that order). That is, SET DATE settings override those in DBASEWIN.INI, and settings in DBASEWIN.INI override those in the Windows Control Panel. Be sure <expD> matches the date format in use when your program runs.

If you want to ensure that records in a table are sorted by date regardless of the current date format in use, and need to concatenate the date field with another field in the table, use DTOS() instead of DTOC(). DTOS() always returns YYYYMMDD, while the value returned by DTOC() depends on the current date format. This means that with DTOS(), dBASE sorts updated records into the table the same way, regardless of the date format in effect. If you use DTOC(), however, the same date format must be current whenever the index is updated; if the date format changes, the index will not be updated correctly.

If you pass an invalid date to DTOS(), dBASE converts the date to a valid one and returns that date as a character string. If you pass an empty or non-date expression delimited with curly braces to DTOS(), it returns an empty string (""). If you pass an expression that isn't delimited with curly braces to DTOS(), it returns an error.

Example

The following examples show the difference between DTOS() and DTOC().

```
SET CENTURY OFF
date1 = {4/1/94}
? DTOC(date1)           && Returns 04/01/94
? DTOS(date1)           && Returns 19940401
SET CENTURY ON
? DTOC(date1)           && Returns 04/01/1994
? DTOS(date1)           && Returns 19940401
SET CENTURY OFF
```

The following example uses DTOS() to convert date field information to a character string with the year first to facilitate searching for a specific year.

```
SET TALK OFF
USE Clients EXCLUSIVE
INDEX ON DTOS(Baldate) TAG Bal_date
yr = "1992"
? CENTER("Report for: "+yr+" - Run on: " + DTOC(DATE()))
?                                     && Spacing line
SCAN FOR SUBSTR(DTOS(Baldate),1,4) = yr
  ? Company + DTOC(Baldate) AT 5
ENDSCAN
SET TALK ON
CLOSE ALL
```

Portability

Not supported in dBASE III PLUS.

See Also

DTOC(), INDEX, SET CENTURY, SET DATE

EDIT**Fields and records**

Displays fields in the current table for editing.

Syntax

```
EDIT
[<starting record expN> | <bookmark>]
[<scope>]
[FOR <condition 1>]
[WHILE <condition 2>]
[COLOR [<standard text>] [, [<enhanced text>] [, <border>]]]
[COLUMNAR]
[FIELDS <field 1> [<field option list 1>] |
  <calculated field 1> = <exp 1> [<calculated field option list 1>]
  [, <field 2> [<field option list 2>] |
  <calculated field 2> = <exp 2> [<calculated field option list 2>]...]]
[FORMAT]
[FREEZE <field 3>]
```

```
[KEY <exp 3> [, <exp 4>]]
[LOCK <expN>]
[NOAPPEND]
[NODELETE]
[NOEDIT | NOMODIFY]
[NOFOLLOW]
[NOINIT]
[NORMAL]
[NOTOGGLE]
[NOWAIT]
[TITLE <expC 1>]
[WIDTH <expN>]
[WINDOW <window name>]
```

<starting record expN> | <bookmark> The record number or bookmark (for tables that don't use record numbers) to edit.

<scope> The number of records to edit. RECORD <n> identifies a single record by its record number. NEXT <n> identifies *n* records, beginning with the current record. ALL specifies all records. REST specifies all records from the current record to the end of the file.

FOR <condition 1>

WHILE <condition 2> Determines which records are affected by EDIT. FOR restricts EDIT to records that meet <condition 1>. WHILE starts processing with the current record and continues with each subsequent record as long as <condition 2> is true.

COLOR Specifies the colors of standard text, enhanced text, and the perimeter of the Table Records window. To specify the colors of these elements separately, use the <standard text>, <enhanced text>, and <perimeter> options. You can also use the <background> option if you have a monitor with a uniform background.

<standard text>	Color attributes of command messages and screen output. For example, the output of the ? and @ ... SAY commands appear in standard text.
<enhanced text>	Color attributes of enhanced text areas, such as @...GET fields and highlighted BROWSE data cells.
<perimeter>	Color attributes of the perimeter bordering the area displaying text on the screen.
<background>	Color attributes of the background for display systems with uniform backgrounds. <background> includes two parameters: a background color and an attribute.

The <standard text> and <enhanced text> attributes include three settings: a foreground color, a background color, and an optional color for creating blended (hatched) backgrounds. Separate each setting with a forward slash (/).

For more information about color settings, see SET COLOR TO and SET COLOR OF.

COLUMNAR Arranges the fields in columns on the editing form.

COMPRESS Reduces the number of rows used to display field names in the Table Records window.

FIELDS <field 1> [<field option list 1>|

<calculated field 1> = <exp 1> [<calculated field option list 1>

[, <field 2> [<field option list 2>|

<calculated field 2> = <exp 2> [<calculated field option list 2>] ...]]

Displays the specified fields, in the order they're listed, for browsing. Options for <field option list 1>, <field option list 2>, which apply to <field 1>, <field 2>, and so on, affect the way these fields are displayed. These options are as follows:

\B = <exp 1>, <exp 2> [\F]

RANGE option; forces any value entered in <field 1> to fall within <exp 1> and <exp 2>, inclusive

RANGE REQUIRED option; the \F option prevents a previously entered value from being accepted if it doesn't fall between <exp 1> and <exp 2>, inclusive

\H = <expC>

HEADER option; causes <expC> to appear above the field column in the Table Records window, replacing the field name

\P = <expC>

PICTURE option; displays <field 1> according to the PICTURE or FUNCTION clause <expC>

\R

READ-ONLY option; specifies that <field 1> is read-only and can't be edited

\V = <condition> [\F] [\E = <expC>]

VALID option; allows a new <field 1> value to be entered only when <condition> evaluates to .T.

VALID REQUIRED option; the \F option prevents the cursor from leaving <field 1> and the editing session from ending until <condition> evaluates to .T.

ERROR MESSAGE option; \E = <expC> causes <expC> to appear when <condition> evaluates to .F.

\W = <condition>

WHEN option; allows <field 1> to be edited only when <condition> evaluates to .T.

Note You may also use the "/" character when specifying only a single option in a field option list.

Read-only calculated fields are composed of an assigned field name and an expression that results in the calculated field value, as with *Commission = Rate * Saleprice*. Options

for calculated fields affect the way these fields are displayed. These options are as follows:

- \<column width> The width of the column within which <calculated field 1> is displayed
- \H = <expC> Causes <expC> to appear above the calculated field column in the Table Records window, replacing the calculated field name

FORMAT Causes EDIT to accept and display input according to the specifications of a format file opened with SET FORMAT. Data entered must conform to any PICTURE, FUNCTION, RANGE, and VALID clauses in the format file.

FREEZE <field 3> Restricts editing to <field 3>, although other fields are visible when you change to browse mode.

KEY <exp 3> [, <exp 4>] When the table has a master index, displays records whose key field value matches or comes after <exp 3>, or falls between <exp 3> and <exp 4>.

LOCK <expN> Causes the first <expN 2> fields in the table, or the first <expN> fields in the field list, to continue to appear in place when you move the cursor to fields to the right in the Table Records window.

NOAPPEND Prevents you from adding records in either browse or edit modes.

NODELETE Prevents the marking of records for deletion from within the Table Records window.

NOEDIT | NOMODIFY Prevents you from modifying records from the Table Records window.

NOFOLLOW When the current table has a master index, causes the cursor to remain in place when you change the key field in a record, rather than follow the record to its new location in the indexing order. Without NOFOLLOW, dBASE re-indexes the table as soon as you move to another record when a master index for the table is open.

NOINIT Causes EDIT to execute the options specified with the previous EDIT command. Use NOINIT if a program calls EDIT several times or if you issue EDIT several times from the Command window, and you want the same options. Specify the command options the first time you use EDIT, and issue EDIT NOINIT for subsequent use in the same session.

NORMAL When EDIT is issued from an active window, displays the Table Records window in normal, full-screen mode with their default or defined colors, ignoring the defined colors of the window. When you exit EDIT, dBASE returns you to the active window. Without NORMAL, the table records appears in the active window.

NOTOGGLE Prevents toggling from EDIT to BROWSE.

NOWAIT Continues execution of a program after a Table Records window is displayed; otherwise, program execution is suspended until after the Table Records window is closed.

TITLE <expC 1> Displays <expC> as the title of the Table Records window.

E

WIDTH <expN> Specifies the display width for character fields in the Table Records window. If a field is wider than the specified width, you can scroll the field within the specified width. The <expN> argument must evaluate to a positive number.

WINDOW <window name> Activates the window <window name> and displays table records in the window.

Description

Use EDIT to display and change the contents of one or more table records. EDIT without the FIELDS option displays all fields of the current record. Use SET RELATION command to view fields from records in linked tables.

To move between records in the Table Records window, you can press *PgUp* and *PgDn* to move backward or forward, one record at a time. You can also use the window controls and the mouse and choose from various menu options to control operations performed with EDIT. See the *User's Guide* for more information on performing operations and navigating in the Table Records window.

When you're through viewing or editing a table in the Table Records window, press *Ctrl+W* to exit EDIT or choose File | Save and Close. To exit EDIT without saving changes to the current record, press *Ctrl+Q*, choose File | Abandon and Close, or double-click the Control menu. If you're using EDIT or BROWSE in a program, exiting EDIT returns program control to the command line immediately following the EDIT command line.

To view and modify a memo field, move the cursor to the memo field and press *Ctrl+Home*, or double-click the memo field. To save the modified memo field, press *Ctrl+W*.

To append a new record (unless you use EDIT...NOAPPEND), move the cursor to the last field of the last record in the table and press *PgDn*.

Many of the EDIT options are the same as those provided by BROWSE. (Press *F2* to toggle between the two modes.)

Use FIELDS followed by a list of table field names or calculated field names to control the fields EDIT and BROWSE display, and whether you can edit data in a particular field.

A calculated field is a read-only field that displays the value of an expression. The expression typically includes one or more fields in the current table, such as

```
Name = Firstname + " " + Lastname
Total = Price + (Salestax * Price)
Monthdue = CMONTH(Billdate + 30)
```

The expression to the left of = specifies the name of the calculated field, and the expression on the right evaluates to each calculated field's contents.

The Table Records window displays fields according to the @...SAY...GET statements of an open format file, whether you use the FORMAT option or not. Use the FORMAT option as a reminder when a format file is open. To open the format file, use SET FORMAT TO and the format file name. If you open a format file, the FIELDS option of EDIT has no effect, and EDIT and BROWSE display only the fields specified in the format file. Although the format file doesn't affect the format of the Table Records

window, data entered in the Table Records window must conform to any PICTURE, FUNCTION, RANGE, or VALID clauses in the format file.

EDIT can alter only the *contents* of a table. To modify the table *structure*, use MODIFY STRUCTURE. EDIT and CHANGE are equivalent commands.

Example

The following example uses EDIT to open a Table Records window containing a subset of the total fields with specified editing limitations:

```
USE Company IN SELECT() EXCLUSIVE
INDEX ON CompCode TAG CompCode
EDIT FIELDS CompCode, Company, ;
    Street1, City, State_Prov, Zip_P_Code ;
NOFOLLOW NOAPPEND NODELETE
CLOSE ALL
```

See Also

APPEND, BROWSE, MODIFY STRUCTURE, SET FORMAT, SET RELATION

E

EJECT

Printing

Advances printer paper to the top of the next page.

Syntax

EJECT

Description

Use EJECT to position printed output on the page. If you are using a tractor-feed printer (such as a dot matrix printer) and the paper is correctly positioned, EJECT advances the paper to the top of the next sheet. If you are using a single-sheet printer (such as a laser printer), EJECT prints any data in the print queue and ejects the page. Before printing or executing EJECT, connect and turn on the printer.

EJECT works in conjunction with `_padvance`, `_plength`, and `_plineno`. If `_padvance` is set to "FORMFEED" (the default), issuing the EJECT command from dBASE is equivalent to using your printer's formfeed button or sending the formfeed character (ASCII 12) to the printer. If `_padvance` is set to "LINEFEEDS", issuing EJECT sends individual linefeeds to the printer until `_plineno` equals `_plength`, then resets `_plineno` to 0. Then, `_pageno` is incremented by 1. For more information, see `_padvance`.

EJECT is often used in when printing reports. For example, if `PROW()` returns a value that is close to the bottom of the page, issue EJECT to continue the report at the top of the next page. EJECT automatically resets the printhead to the top left corner of the new page, which is where `PROW() = 0` and `PCOL() = 0`.

EJECT is the same as EJECT PAGE, except EJECT PAGE also executes any page-handling routine you've defined with ON PAGE.

Example

In this example, one line is written to the printer and EJECT is then used to ensure that further commands are written on the next page:

```
SET PRINTER ON
? "This page left intentionally blank"
EJECT
```

See Also

`_padvance`, `_plength`, `_plineno`, EJECT PAGE, ON PAGE, PCOL(), PROW(), SET PRINTER, SET PROW

EJECT PAGE

Printing

Advances printer paper to the top of the next page and executes any ON PAGE command.

Syntax

EJECT PAGE

Description

Use EJECT PAGE with ON PAGE to control the ejection of pages by a printer. If you define a page-handling routine with ON PAGE AT LINE *<expN>* and then issue EJECT PAGE, dBASE checks to see if the current line number (`_plineno`) is greater than the line number specified by *<expN>*. If `_plineno` is less than the ON PAGE line, EJECT PAGE sends sufficient linefeeds to trigger the ON PAGE page-handling routine.

If `_plineno` is greater than the ON PAGE line, or if you don't have an ON PAGE page-handling routine, EJECT PAGE advances the output as follows:

- If `_padvance` is set to "FORMFEED" and SET PRINTER is ON, dBASE issues a formfeed (ASCII code 12).
- If `_padvance` is set to "LINEFEEDS" and SET PRINTER is ON, dBASE issues sufficient linefeeds (ASCII code 10) to advance to the next page. It uses the formula `_plength - _plineno` to calculate the number of linefeeds.
- If you direct output to a destination other than the printer (for example, if you use SET ALTERNATE or SET DEVICE), dBASE uses the formula `_plength - _plineno` to calculate the number of linefeeds.

After ejecting a page, EJECT PAGE increments `_pageno` by 1 and resets `_plineno` to 0.

Example

See ON PAGE for an example of EJECT PAGE.

Portability

Not supported in dBASE III PLUS.

See Also

?, ??, _padvance, _pageno, _plength, _plineno, EJECT, ON PAGE, PRINTJOB...ENDPRINTJOB, SET ALTERNATE, SET DEVICE, SET PRINTER, SET PROW

ELAPSED()**Date and time data****E**

Returns the number of seconds elapsed between two specified times.

Syntax

ELAPSED(<stop time expC>, <start time expC>)

<stop time expC> The time expression, in the format HH:MM:SS, at which to stop timing seconds elapsed. The <stop time expC> argument should be a later time than <start time expC>; if it is not, dBASE returns a negative value.

<start time expC> The time expression, in the format HH:MM:SS, at which to start timing seconds elapsed. The <start time expC> argument should be an earlier time than <stop time expC>; if it is not, dBASE returns a negative value.

Description

Use ELAPSED() with TIME() to determine benchmarks for a program—that is, to determine how long a program takes to execute when you use different programming strategies. You can also use ELAPSED() to determine a particular computer's processing speed; you can then slow down screen displays that would otherwise execute too quickly. You can also use ELAPSED() to design timeout features in automated demonstration programs and tutorials.

ELAPSED() subtracts the value of <start time expC> from <stop time expC>. If <start time expC> is the later time, ELAPSED() returns a negative integer. Both <stop time expC> and <start time expC> must be in HH:MM:SS format, where HH is the hour, MM the minutes, and SS the seconds; however, you don't need to specify minutes and seconds if you want to designate them as 0.

Example

The following example uses ELAPSED() to compute elapsed time in hours between two entered times:

```
LOCAL f
f = NEW GFORM()
f.Open()

CLASS GFORM OF FORM
    this.Left = 53.00
    this.Height = 15.00
    this.Width = 43.00
    this.Text = "Time worked today?"
    this.HelpId = ""
    this.HelpFile = ""
    this.Top = 5.59
```

ELAPSED()

```
DEFINE TEXT T1 OF THIS;
PROPERTY;
  Left      4.00;;
  Height    1.00;;
  ColorNormal "N/W",;
  Width     28.00;;
  Border .F.,;
  Text "Enter Start Time (24 hr):",;
  Top       3.00

DEFINE ENTRYFIELD F1 OF THIS;
PROPERTY;
  Left      34.00;;
  Height    1.00;;
  Width     5.00;;
  Value "   : ",;
  Picture "99:99",;
  Border .T.,;
  Top       3.00

DEFINE TEXT T2 OF THIS;
PROPERTY;
  Left      4.00;;
  Height    1.00;;
  ColorNormal "N/W",;
  Width     28.00;;
  Border .F.,;
  Text "Enter Quit Time (24 hr):",;
  Top       5.00

DEFINE ENTRYFIELD F2 OF THIS;
PROPERTY;
  Left      34.00;;
  Height    1.00;;
  Width     5.00;;
  Value "   : ",;
  Picture "99:99",;
  Border .T.,;
  Top       5.00

DEFINE PUSHBUTTON COMPUTE OF THIS;
PROPERTY;
  Left      12.00;;
  Height    2.00;;
  ColorNormal "N/W",;
  Width     19.00;;
  OnClick {;myResult="Time worked today: " +
LTRIM(STR(ELAPSED(Form.F2.Value, Form.F1.Value)/3600,6,2)) + " Hours"
; Form.WorkTime.Text=myResult},;
  Text "Elapsed Time",;
  Default .T.,;
  Top       9.00

DEFINE TEXT WORKTIME OF THIS;
PROPERTY;
  Left      4.00;;
  Height    1.00;;
```

```

ColorNormal "R+/W",;
Width      30.00;;
Border .F.,;
Text "Time worked today: ",;
Top      7.00

```

```
ENDCLASS
```

Portability

Not supported in dBASE IV or dBASE III PLUS.

E

See Also

SECONDS(), TIME()

EMPTY()

Expressions and type conversion

Returns .T. if a specified expression or field is 0 or blank, .F. if it contains any other value.

Syntax

EMPTY(<exp>)

<exp> An expression of any data type.

Description

EMPTY() returns .T. if a specified expression is blank or 0, .F. if it contains data or a numeric value other than 0.

EMPTY() is almost identical to ISBLANK(). However, ISBLANK() differentiates between zero and blank values in numeric fields, while EMPTY() does not. For more information, see ISBLANK().

Example

The following example compares the use of AVERAGE when used with no conditions, with ISBLANK(), and with EMPTY().

File Blnktest.DBF contains 4 records, with the following values:

Record #	NUMFIELD
1	0
2	10
3	20
4	(blank)

The following includes all records, so it does not calculate a true average if blank values can be included.

```
AVERAGE numfield TO nAver1    && Returns 7.5 (30/4)
```

The following ignores blank records, but includes those with a value of 0. This is the most accurate average if 0 values are valid.

EOF()

```
AVERAGE numfield TO nAver2 ;  
FOR .NOT. ISBLANK(numfield)    && Returns 10 (30/3)
```

The following ignores records that are blank or have a value of 0. Use this if you want to exclude records with no value, or with a value of 0.

```
AVERAGE numfield TO nAver3 ;  
FOR .NOT. EMPTY(numfield)    && Returns 15 (30/2)
```

Portability

Not supported in dBASE III PLUS or dBASE IV.

See Also

BLANK, ISBLANK(), SPACE(), TYPE()

EOF()

Fields and records

Indicates if the record pointer is at the end of the file.

Syntax

EOF([<alias>])

<alias> A work area number (1 through 225), letter (A through J), or alias name. The work area letter or alias name must be enclosed in quotes.

Description

EOF() returns .T. when the record pointer in the current or specified table is positioned past the last record or when the record pointer is positioned past the last record in the master index file. Otherwise, EOF() returns .F. Also, if no table is open in the specified work area, EOF() returns .F.

EOF() returns .T. after SCAN processes the last record in a table, when you use SKIP to pass the last record in a table or index file, when you use LIST with no options, or when CONTINUE, FIND, GO, LOCATE, LOOKUP(), SEEK(), or SEEK fails to find the specified record (and SET NEAR is OFF).

Example

The following example uses EOF() to test for the end of file during a DO...UNTIL loop. When the record pointer skips past the last record, EOF() becomes true and the DO...UNTIL loop ends:

```
USE COMPANY  
DO  
  * ...  
  SKIP                && skip to next record  
UNTIL EOF()           && test for EOF()
```

The following example defines a form and two pushbuttons for moving the record pointer. BOF() and EOF() are used to avoid a BOF() or EOF() error alert at either end of the table:

```

SET PROCEDURE TO PROGRAM(1) ADDITIVE
USE Clients
DEFINE FORM F1
DEFINE PUSHBUTTON Pb1 OF F1 AT 10,10;
    PROPERTY Text "Previous", Width 8, OnClick Back
DEFINE PUSHBUTTON Pb2 OF F1 AT 10,22;
    PROPERTY Text "Next", Width 8, OnClick Forward
OPEN FORM F1

FUNCTION Back
IF .NOT. BOF()
    SKIP-1
ENDIF
RETURN .T.

FUNCTION Forward
IF .NOT. EOF()
    SKIP
ENDIF
RETURN .T.

```

E

See Also

BOF(), FIND, FOUND(), LOCATE, RECNO(), SEEK, SEEK()

ERASE

Disk and file utilities

Removes a file from a disk.

Syntax

ERASE <filename> | ? | <filename skeleton>

<filename> | ? | <filename skeleton> Identifies the file to remove. ? and <filename skeleton> display a dialog box from which you can select a file.

If you specify a file without including its path, dBASE looks for the file in the current directory, then in the path you specify with SET PATH. If you specify a file without including its extension, dBASE does not assume an extension.

Description

ERASE is a utility command that deletes a file from a disk. ERASE and DELETE FILE are equivalent commands.

ERASE is nearly identical to the ERASE (or DEL) command in DOS. Unlike the DOS command, however, including the wildcard characters * and ? in a file-name skeleton opens a dialog box from which to choose file names to erase. Also unlike the DOS command, you can delete only one file at a time with ERASE. A file must be closed before ERASE can remove it.

If the file that you want to remove has an extension, you must include that extension in <filename>. If the file is not on the default drive you must specify a drive designation,

ERROR()

and if the file is not in the current directory or in the path you specify with SET PATH, you must specify the directory path.

If <filename> is present in the current directory and also exists in the SET PATH directory, ERASE <filename> without path information removes only the file in the current directory. SET SAFETY has no effect on ERASE.

ERASE does not automatically remove a .DBT file when the .DBF file it is associated with is removed. To remove all files associated with a table, use DELETE TABLE.

Example

The following examples use ERASE:

```
ERASE Temp.prg
ERASE ?
* Displays the open source dialog box
```

Portability

The <filename skeleton> argument is not supported in dBASE IV or dBASE III PLUS.

See Also

DELETE TABLE, RENAME, SET DIRECTORY, SET PATH, SET SAFETY

ERROR()

Error handling and debugging

Returns the number of the most recent dBASE error.

Syntax

ERROR()

Description

Use ERROR() to determine the error number when an error occurs. ERROR() is initially set to 0. ERROR() returns an error number when an error occurs, and remains set to that number until one of the following happens:

- Another error occurs
- RETRY is issued
- The subroutine in which the error occurs completes execution

The following table compares the functionality of CERROR(), DBERROR(), DBMESSAGE(), ERROR(), MESSAGE(), SQLERROR(), and SQLMESSAGE().

Function	Returns
CERROR()	Compiler error number
DBERROR()	IDAPI error number
DBMESSAGE()	IDAPI error message
ERROR()	dBASE error number

Function	Returns
MESSAGE()	dBASE error message
SQLERROR()	Server error number
SQLMESSAGE()	Server error message

See online Help for a listing of all error codes.

Example

See ON ERROR for an example of using ERROR().

Portability

The error codes of some errors in dBASE IV and dBASE III PLUS are different from the error codes for the same errors in *Visual* dBASE. See online Help for more information.

An ON ERROR command does not need to be active in *Visual* dBASE for ERROR() to return an error code.

See Also

CERROR(), DBERROR(), DBMESSAGE(), MESSAGE(), ON ERROR, RETRY, SQLERROR(), SQLMESSAGE()



EXP()

Numeric data

Returns *e* raised to a specified power.

Syntax

EXP(<expN>)

<expN> The positive, negative, or zero power (exponent) to raise the number *e* to.

Description

EXP() returns a float equal to *e* (the base of the natural logarithm) raised to the <expN> power. For example, EXP(2) returns 7.39 because *e*²=7.39.

EXP() is the inverse of LOG(). For example, if Y=EXP(X), then LOG(Y)=X.

Use SET DECIMALS to set the number of decimal places EXP() displays.

Example

The following example uses EXP() to determine the exponential of several values:

```
SET DECIMALS TO 6
? EXP(1)           && Returns 2.718282;
                   the value of e
? EXP(0)           && Returns 1.000000
? EXP(-1)          && Returns 0.367879
? EXP(-10)         && Returns .000045
? EXP(-50)         && Returns 0.000000
```

The following example shows how to calculate 25*25 using *e* and natural logarithms:

```
dec = SET("DECIMALS")
SET DECIMALS TO 2
x = 25
y = LOG(x) + LOG(x) && y = 6.44
? EXP(y)           && Returns 625.00
SET DECIMALS TO dec
```

See Also
LOG(), LOG10(), SET DECIMALS

EXTERN

Windows programming

Declares a prototype for a non-dBASE function contained in a DLL file.

Syntax
EXTERN [CDECL] <return type> <function name>
([<parameter type list>])
<path> <filename>

or
EXTERN [CDECL] <return type> <user-defined function name>
([<parameter type list>])
<path> <filename>
FROM <export function name> | <ordinal number>

Since you create a function prototype with EXTERN, parentheses are required as with other functions. Parentheses affect the way data types are promoted and converted.

CDECL Makes EXTERN use the C calling convention. If you omit CDECL, dBASE uses the Pascal calling convention. (See the following table.)

<function name> The export name of the function. The export name of an external function is contained in the .REF file associated with the DLL file that holds the function.

<return type> and <parameter type> A keyword representing the data type of the value returned by the function, and the data type of each argument you send to the function, respectively. The following table lists the keywords you can use.

Keyword	dBASE data type	C data type	Pascal data type	ASM data type
Parameters or return values				
CDOUBLE	Numeric	long double (80 bit)	Double	N/A
CHANDLE	Numeric	Handles, such as HANDLE, HWND, HFONT, HDC	Handles, such as Hwnd, HFont, HDC	dw
CINT	Numeric	int	Integer	dw (16 bit)
CLOGICAL	Logical	short Int	Integer	dw (16 bit)
CLONG	Numeric	long int (32 bit)	Long Int	dd (32 bit)

Keyword	dBASE data type	C data type	Pascal data type	ASM data type
CSTRING	Character	char far * (zero terminated)	PChar	dw (16 bit)
CVOID	N/A	void	Procedure	N/A
CWORD	Numeric	short int (16 bit)	WORD	dw (16 bit)
Parameters only				
CPTR	N/A	void *	Pointer	dd (32 bit)

E

For example, a C function may expect a 32-bit unsigned long value as a parameter, and return a char * string. In your EXTERN command, you specify CLONG as the parameter in *<parameter type list>* and CSTRING as the *<return type>*. When you call the function, you pass a dBASE numeric variable and store the returned value to a character variable.

<user-defined function name> The name you give to the external function instead of the export name. When you specify *<user-defined function name>* (instead of *<function name>*), you must use the FROM *<expC>* | *<expN>* clause to identify the function in the DLL file.

FROM <export function name> | <ordinal number> Identifies the function in the DLL file specified by *<filename>*. *<export function name>* identifies the function by its name, which is stored in the .DEF file that is associated with the DLL file. *<ordinal number>* identifies the function with a number, which is also stored in the .DEF file.

When the function you call does not return a value, specify CVOID for *<return type>*.

<filename> The name of the DLL file in which the external function is stored. This name must include the extension if the DLL file is not already in memory. The file name of any DLL that you load in memory must be unique; for example, you can't load SCRIPT.DLL and SCRIPT.FON into memory concurrently, even though they have different file-name extensions.

If the DLL file is not already loaded into memory, EXTERN loads it automatically. If the DLL file is already in memory, EXTERN increments the reference counter once. Therefore, it isn't necessary to execute LOAD DLL before using EXTERN.

The reference counter is incremented only the first time, regardless of how many times you execute the LOAD DLL and EXTERN commands.

<path> The directory path to the DLL file in which the external function is stored. When you omit *<path>*, dBASE looks in the following directories for the DLL by default:

- 1 The current directory.
- 2 The Windows directory (for example, C:\WINDOWS).
- 3 The Windows SYSTEM subdirectory (for example, C:\WINDOWS\SYSTEM).
- 4 The directory containing DBASEWIN.EXE, or the directory in which the .EXE file of your compiled program is located.
- 5 The directories in the current DOS path.
- 6 The directories mapped for search in a network.

The *<path>* specification is necessary only when the DLL file is not in one of these directories.

Description

Use EXTERN to declare a prototype for an external function written in a language other than dBASE. A prototype tells dBASE to convert its arguments to data types the external function can use, and to convert the value returned by the external function into a data type dBASE can use.

To call an external DLL function, first prototype it with EXTERN. Then, using the name of the function you specified with EXTERN, call the function as you would any dBASE function. You must prototype an external function before you can call that function in your dBASE program.

The external function is held in a C library such as Windows API or a customized DLL file you create in C, Pascal, or ASM. (For more information on using EXTERN and DLL files, see Chapter 25 in the *Programmer's Guide*.) Although most library code is contained in files with extensions of .DLL, such code can be held in .EXE files, or even in .DRV or .FON files.

Example

The following example is a dBASE program that uses EXTERN to call a C program (cSample1.PRG). Explanations of included functions are as follows:

```
* Reverse():
*   Uses the FROM option to specify a new name for
*   the function (StrRevC) in the .DLL that has the
*   ordinal number 2. In this example the var
*   myString is passed by reference, it is then
*   modified by the function StrRevC() in the dll
*   and the modified string is then displayed in
*   dBASE.
* GetDOSEnv():
*   This example shows returning a address of a
*   String to display in dBASE from a .dll, the
*   String is one of the DOS Environment strings.
* GetDOSEnvNum():
*   This example shows returning a number to dBASE
*   from a .dll, the number is the number of null
*   terminated strings in the DOS Environment. Use
*   this first to find how many strings to get from
*   GetDOSEnv().
* GetDOSEnvLen():
*   This example shows returning a number to dBASE
*   from a .dll, the number is the length of all the
*   strings in the DOS Environment.
*****
CLEAR
EXTERN CVOID   Reverse(CSTRING) cSample1.dll FROM 2
* 2 is the ORDINAL number in the DLL cSample.dll
EXTERN CSTRING GetDOSEnv(CWORD) cSample1.dll
EXTERN CWORD   GetDOSEnvLen()   cSample1.dll
EXTERN CWORD   GetDOSEnvNum()   cSample1.dll
myString = "AbCdE"
? "Original Str: ",myString
Reverse(myString)
? "From StrRevC: ",myString
```

```

nSize=GetDOSEnvLen()
? "The Size of the DOS Environment Sting is: ",nSize
?
? "The DOS Environment is:"
FOR i = 1 to GetDOSEnvNum()    && for 1 to Number of;
                                Strings in DOS Env
    ? GetDOSEnv(i)            && print each String;
                                from the DOS Env
NEXT

```

E

The C source code for creating the .DLL file is as follows: (cSample1.C)

```

#include <windows.h>
#include <string.h>
#pragma argsused
int FAR PASCAL LibMain(HINSTANCE hInstance,
    WORD wDataSeg, WORD wHeapSize, LPSTR lpszCmdline){
    if (wHeapSize > 0) UnlockData (0) ;
    return 1;}
#pragma argsused
int FAR PASCAL WEP(int wParameter) {
    return 1;}
/* #####
  ## Function StrRevC()          ##
  #####*/
#pragma argsused
void FAR PASCAL StrRevC(LPSTR lpstrString) {
    strrev(lpstrString);}
/* #####
  ## Functions related to  GetDOSEnv() ##
  #####*/
#pragma argsused
LPSTR FAR PASCAL GetDOSEnv(UINT index) {
    LPSTR p;
    UINT i;
    for(i=1,p=GetDOSEnvironment();
        i<index && *p != '\0';
        p += (lstrlen(p)+1),++i);
    return p;}
#pragma argsused
UINT FAR PASCAL GetDOSEnvLen(void) {
    return (UINT)
        (GetDOSEnv((UINT)-1) - GetDOSEnv(1)) + 1;}
#pragma argsused
UINT FAR PASCAL GetDOSEnvNum(void) {
    LPSTR p;
    UINT i;
    for(i=0,p=GetDOSEnvironment(); *p != '\0';
        p += (lstrlen(p)+1),++i);
    return i;}

```

The C source code for creating the .DEF file is as follows(cSample1.DEF):

```

LIBRARY      CSAMPLE1
DESCRIPTION  'A Sample DLL for dBASE for Windows'
EXETYPE      WINDOWS
CODE         PRELOAD MOVEABLE DISCARDABLE

```

EXTERN

```
DATA          PRELOAD SINGLE
HEAPSIZE      1400

EXPORTS
              WEP                @1
              STRREVC            @2
              GETDOENV           @3
              GETDOENVLEN        @4
              GETDOENVNUM        @5
```

The following section demonstrates the FROM option of EXTERN (using the name of the function in the .DLL): This code segment displays a messagebox with an 'i' icon, title of 'MyBox', message of 'Hello World', and 2 buttons, 'OK' and 'Cancel'.

```
EXTERN CWORD MyBox(CHANDLE,CSTRING,CSTRING,CWORD);
user.exe FROM "MessageBox"
? MyBox(0,"Hello World","MyBox",65)
```

Example 2: Calling a PASCAL dll. In this example the variable myString is passed by reference. It is then modified by the function StrRevP() in the dll and the modified string is returned.

```
* dBASE Program (pStrRev.PRG)
EXTERN CVOID StrRevP(CSTRING) pStrRev.dll
myString = "ABCD"
StrRevP(myString)
? myString
```

DLL Source Code (pStrRev.PAS)

```
Library pStrRev;
Uses
  Strings;
Function StrRevP(str : PChar) : PChar; Export;
Var
  endstr : PChar;
  ch      : Char;
Begin
  StrRevP := str;
  If Assigned(str) And (str^ <> #0) Then
    Begin
      endstr := StrEnd(str);
      Dec(endstr);
      While endstr > str Do
        Begin
          ch := str^;
          str^ := endstr^;
          endstr^ := ch;
          Inc(str);
          Dec(endstr)
        End
      End
    End;
Exports
  StrRevP Index 1;
Begin
End.
```

Portability

Not supported in dBASE IV or dBASE III PLUS.

See Also

LOAD DLL, RELEASE DLL

FACCESSDATE()**Windows 95**

Returns the last date a file was opened under Windows 95.

F**Syntax**

FACCESSDATE(<filename expC>)

<filename expC> The name of the file to check. If the file is not in the current directory, you must provide the path. If the file does not exist, dBASE returns an error message.

Description

FACCESSDATE() checks the file specified by <filename> and returns the date that the file was last opened, provided the file was last opened under the Windows 95 operating system. This function is only useful on a system running the Windows 95 operating system. Under Windows 3.1, FACCESSDATE() returns a blank date.

Example

The following example uses FACCESSDATE() to check the last opened date of a table:

```
? FACCESSDATE("C:\VISUALDB\SAMPLES\ANIMALS.DBF")
06/01/95
```

Portability

Not supported in dBASE IV or dBASE III PLUS.

See Also

FCREATEDATE(), FCREATETIME(), FDATE(), FTIME()

FCLOSE()**Low-level access**

Closes a file previously opened with FCREATE() or FOPEN(). Returns .T. if successful, .F. if unsuccessful.

Syntax

FCLOSE(<file handle expN>)

<file handle expN> The *file handle number* of the file to close. When you open a file with FCREATE() or FOPEN(), these functions return a file handle number. Use this number as <file handle expN>. If you specify a file handle number that hasn't previously been returned by FCREATE() or FOPEN(), dBASE returns an error message.

FCREATE()

Description

FCLOSE() closes a file you've opened with FCREATE() or FOPEN(). FCLOSE() returns .T. if it's able to close the file. If the file is no longer available (for example, the file was on a floppy disk that has been removed) and there is data in the buffer that has not yet been written to disk, FCLOSE() returns .F.

FCLOSE() doesn't close files that were opened by any means other than FCREATE() or FOPEN().

To close all open files, including those opened with FCREATE() and FOPEN(), use CLOSE ALL or CLEAR ALL. To save the file to disk without closing it, use FFLUSH().

Example

The following example uses FCLOSE() to close the README.TXT file after adding text:

```
IF FILE("C:\VISUALDB\README.TXT")
  Handle=FOPEN("README.TXT","RW")    && Opens Read-Write
  FSEEK(Handle,0,2)                    && Move to EOF
  FPUTS(Handle,"The End")              && Write to file
  IF .NOT. FCLOSE(Handle)              && Try closing file
    ? "Couldn't close the file"
    ? "The DOS error number is:", FERROR()
  ENDEF
ENDEF
```

Portability

Not supported in dBASE III PLUS.

See Also

CLEAR ALL, CLOSE..., FCREATE(), FERROR(), FFLUSH(), FOPEN()

FCREATE()

Low-level access

Creates and opens a specified file. Returns the *file handle number* of the file if successful or -1 if unsuccessful.

Syntax

FCREATE(<filename expC>[, <access expC>])

<filename expC> The name of the file to create and open and whose file handle number to return. By default, FCREATE() creates the file in the current directory. To create the file in another directory, specify a full path name for <filename expC>.

<access expC> The access level of the file to create, as shown in the following table. *Write* means you can change (overwrite) data in the file, and *append* means you can add data to the end of the file. If you try to overwrite data in a file that has append access but not write access, the data is added to the end of the file.

<access expC>	Access level
<i>not supplied</i>	Read, write, and append
"R"	Read-only
"W"	Write-only
"A"	Append-only
"RW" or "WR"	Read and write
"AR" or "RA"	Read and append
"AW" or "WA"	Write and append

Description

Use FCREATE() to create a file with a name you specify, assign the file the level of access you specify, open the file, and return the file handle number DOS assigns to the file. If dBASE can't create the file (for example, if the file is already open), FCREATE() returns -1.

SET SAFETY has no effect on FCREATE(). If <filename expC> already exists, it is overwritten without warning. To see if a file with the same name already exists, use FILE() before issuing FCREATE().

To use other low-level functions, such as FREAD() and FWRITE(), first open a file with FCREATE() or FOPEN(). Both FCREATE() and FOPEN() return the file handle number you need to pass to other low-level functions.

When you open a file with FCREATE(), the file pointer is positioned at the first character in the file. Use FSEEK() to position the file pointer before reading from or writing to a file.

Example

The following example uses FCREATE() to create a file named JACK.TXT with Read and Append access levels:

```
Filename="Jack.TXT"
? FILE(Filename)    && Returns .F.;
                    file does not currently exist
Handle=FCREATE(Filename,"RA")
IF Handle>0
  FPUTS(Handle,"This is a test file.")
  FCLOSE(Handle)
  FOPEN(Filename)
  ? FGETS(Handle)    && Returns "This is a test file."
ENDIF
WAIT
CLEAR ALL
```

Portability

Not supported in dBASE III PLUS.

See Also

FCLOSE(), FERROR(), FGETS(), FILE(), FOPEN(), FREAD(), FSEEK(), SET ALTERNATE, SET DEVICE, SET SAFETY

FCREATEDATE()**Windows 95**

Returns the date a file was created under Windows 95.

Syntax

FCREATEDATE(<filename expC>)

<filename expC> The name of the file to check. If the file is not in the current directory, you must provide the path. If the file does not exist, dBASE returns an error message.

Description

FCREATEDATE() checks the file specified by <filename> and returns the date that the file was created, provided the file was created under the Windows 95 operating system. This function is only useful on a system running the Windows 95 operating system. Under Windows 3.1, FCREATEDATE() returns a blank date.

Example

The following example uses FCREATEDATE() to check the creation date of a table:

```
? FCREATEDATE("C:\VISUALDB\SAMPLES\ANIMALS.DBF")
06/01/95
```

Portability

Not supported in dBASE IV or dBASE III PLUS.

See Also

FACCESSDATE(), FCREATETIME(), FDATE(), FTIME()

FCREATETIME()**Windows 95**

Returns the time a file was created under Windows 95.

Syntax

FCREATETIME(<filename expC>)

<filename expC> The name of the file to check. If the file is not in the current directory, you must provide the path. If the file does not exist, dBASE returns an error message.

Description

FCREATETIME() checks the file specified by <filename> and returns the time, as a character string, that the file was created, provided the file was created under the Windows 95 operating system. This function is only useful on a system running the

Windows 95 operating system. Under Windows 3.1, FCREATETIME() returns an empty string.

Example

The following example uses FCREATEDATE() to check the creation time of a table:

```
? FCREATETIME("C:\VISUALDB\SAMPLES\ANIMALS.DBF")
12:20:10
```

Portability

Not supported in dBASE IV or dBASE III PLUS.

F

See Also

FACCESSDATE(), FCREATEDATE(), FDATE(), FTIME()

FDATE()

Disk and file utilities

Returns the date stamp for the specified file.

Syntax

FDATE(<filename expC>)

<filename expC> The name of the file to evaluate. Wildcard characters are not supported.

If you specify a file without including its path, dBASE looks for the file in the current directory, then in the path you specify with SET PATH. If you specify a file without including its extension, dBASE does not assume an extension.

Description

Use FDATE() to determine the date on which the last change was made to a file on disk.

When you update a file, dBASE changes the file's date stamp to the current operating system date when the file is written to disk. For example, when the user edits a table, dBASE changes the date stamp on the table file when the file is closed. FDATE() reads the date stamp and returns its current value.

If the file that you want to evaluate has an extension, you must include that extension in <filename expC>. If the file is not on the default drive you must specify a drive designation, and if the file is not in the current directory or in the path you specify with SET PATH, you must specify the directory path.

If dBASE cannot find the file, it returns an error. Therefore, you may want to test for its existence with FILE() before issuing FDATE(). FLUSH does not update a file's time stamp.

If <filename expC> is present in the current directory and also exists in the SET PATH directory, FDATE(<filename expC>) (without path information) returns information on the file in the current directory.

Example

This example compares the date and time stamps on COMPANY.DBF and its backup on the B drive. If the backup is dated earlier than the current table then a backup is recommended:

```
IF FDATE("B:Company.dbf") < FDATE("Company.dbf");
  .AND.;
  FTIME("B:Company.dbf") < FTIME("Company.dbf")
  ? "Time to do a backup"
Difference= ;
  FDATE("Company.dbf") - FDATE("B:Company.dbf")
  ? Difference, " days since last backup"
WAIT
ENDIF
```

Portability

Not supported in dBASE III PLUS.

See Also

FILE(), FLUSH, FSIZE(), FTIME(), SET DIRECTORY, SET PATH

FDECIMAL()

Fields and records

Returns the number of decimal places in a specified field of a table.

Syntax

FDECIMAL(<field number expN> [, <alias>])

<field number expN> The position of the field that you want to evaluate.

<alias> A work area number (1 through 225), letter (A through J), or alias name. The work area letter or alias name must be enclosed in quotes.

Description

FDECIMAL() returns the number of decimal places in a specified field of a table. FDECIMAL() returns zero if the field has no decimal places or if the table doesn't contain a field in the specified position. FDECIMAL() returns an error if you specify a non-numeric field.

Fields in the table are numbered from 1 to 1024 based on their position in the table structure. If you do not specify a work area, FDECIMAL() evaluates the table in the current work area.

Example

The following example uses FDECIMAL() to return the number of decimal places in the numeric field StartBal of the Clients table:

```
CLEAR
SET TALK OFF
USE Clients
```

```

DISPLAY STRUCTURE      && StartBal is field number 11
STORE FDECIMAL(11,1) TO m_places
? FIELD(11) + " has " + LTRIM(STR(m_places)) + ;
  " decimal places."    && Returns a value of 2
SET TALK ON

```

See FLENGTH() for another example of FDECIMAL().

Portability

Not supported in dBASE IV and dBASE III PLUS.

See Also

FIELD(), FLENGTH()

F

EOF()

Low-level access

Returns .T. if the file pointer is at the end of a file previously opened with FCREATE() or FOPEN().

Syntax

EOF(<file handle expN>)

<file handle expN> The *file handle number* of the file whose file-pointer position to determine. When you open a file with FCREATE() or FOPEN(), these functions return a file handle number. Use this number as <file handle expN>. If you specify a file handle number that hasn't previously been returned by FCREATE() or FOPEN(), dBASE returns an error message.

Description

EOF() determines if the file pointer of the file you specify is at the end of the file (EOF), and returns .T. if it is. The file pointer is considered to be at EOF if it is positioned at the byte after the last character in the file.

You can move the file pointer to the end of the file with FSEEK(). For example, FSEEK(fnum,0,2) moves the file pointer to the end of the file whose file handle number is stored in the variable fnum.

Example

The following example opens the README.TXT file and displays or prints the file one line at a time until reaching the end of the file:

```

SET PATH TO C:\VISUALDB
Handle=FOPEN("Readme.TXT","R")    && Opens, read only
SET PRINTER ON                     && optional
DO WHILE .NOT. EOF(Handle)         && loops until EOF
  ? FGETS(Handle)                  && sends 1 line to display or printer
ENDDO
SET PRINTER OFF

```

Portability

Not supported in dBASE III PLUS.

See Also

FCLOSE(), FCREATE(), FERROR(), FOPEN(), FSEEK()

FERROR()**Low-level access**

Returns the error number of the most recent low-level input or output error, or 0 if the most recent low-level function was successful.

Syntax

FERROR()

Description

Use the number that FERROR() returns in an error-handling routine to respond to low-level errors. The following table lists the low-level function errors that FERROR() returns.

Error number	Cause of error
2	File or directory not found
3	Bad path name
4	No more file handle numbers available
5	Can't access file
6	Bad file handle
8	No more directory entries available
9	Error trying to set the file pointer
13	No more disk space
14	End of file

Example

The following example attempts to open a file with FOPEN() when the file name is misspelled. The IF/ELSE branch uses FERROR() to return the DOS error message that was returned when FOPEN() failed:

```
SET PATH TO C:\VISUALDB
Filename="Reedme.TXT"
* Filename misspelled
IF FOPEN(Filename) = -1
* Returns -1 if open or not successful
? "The DOS error number is: ", LTRIM(STR(FERROR()))
* FERROR() returns 2 - File or directory not found
ENDIF
RETURN
```

Portability

Not supported in dBASE III PLUS.

See Also

FCLOSE(), FCREATE(), FEOF(), FFLUSH(), FGETS(), FOPEN(), FPUTS(), FREAD(), FSEEK(), FWRITE(), ON ERROR

FFLUSH()**Low-level access****F**

Writes to disk a file previously opened with FCREATE() or FOPEN(), without closing the file. Returns .T. if successful, .F. if unsuccessful.

Syntax

FFLUSH(<file handle expN>)

<file handle expN> The *file handle number* of the file to write to disk. When you open a file with FCREATE() or FOPEN(), these functions return a file handle number. Use this number as <file handle expN>. If you specify a file handle number that hasn't previously been returned by FCREATE() or FOPEN(), dBASE returns an error message.

Description

Use FFLUSH() to save a file in the file buffer to disk, flush the file buffer, and keep the file open. If FFLUSH() is successful, it returns .T.

Flushing a buffer to disk is similar to saving the file and continuing to work on it. Until you flush an open file buffer to disk, any data in the buffer is stored only in RAM (random-access memory). If the power to the computer fails or dBASE ends abnormally, data in RAM is lost. However, if you have used FFLUSH() to write the file buffer to disk, you lose only data that was added between the time you issued FFLUSH() and the time the system failed.

To save the file to disk and close the file, use FCLOSE().

Example

The following example creates a file named JACK.TXT, inputs a text string into the file and writes to disk with FFLUSH() without closing the file. FPUTS() adds additional text which is subsequently displayed with the DO WHILE .NOT. FEOF() loop:

```
Handle=FCREATE("Jack.TXT", "RW")
Filename="Jack.Txt "
FPUTS(Handle, "dBASE means")      && Text to file
FFLUSH(Handle)                   && Saved to disk file still open
FSEEK(Handle, 0, 0)               && Move pointer to the beginning of file
? FGETS(Handle)                   && Displays text
WAIT                             && Pauses program
CLEAR                             && Clears screen
FPUTS(Handle, "Quality in Software Craftsmanship") && Appends text
FCLOSE(Handle)                   && Close Jack.TXT
Handle =FOPEN(Filename, "R")      && Reopen file
DO WHILE .NOT. FEOF(Handle)
```

FGETS()

```
? FGETS(Handle)           && Display contents
ENDDO
RETURN
```

Portability
Not supported in dBASE III PLUS.

See Also
FCLOSE(), FCREATE(), FEOF()

FGETS()

Low-level access

Returns a character string from a file previously opened with FCREATE() or FOPEN().

Syntax
FGETS(<file handle expN> [, <characters expN>] [, <end-of-line exp>])

<file handle expN> The *file handle number* of the file whose characters to read and return. When you open a file with FCREATE() or FOPEN(), these functions return a file handle number. Use this number as <file handle expN>. If you specify a file handle number that hasn't previously been returned by FCREATE() or FOPEN(), dBASE returns an error message.

<characters expN> The number of characters to read and return before a carriage return is reached. Acceptable values are 0 to 32766; if <characters expN> is less than 0 or greater than 32766, dBASE uses 0 or 32766, respectively.

<end-of-line exp> The end-of-line indicator, which can be one or two characters. The following table lists standard codes for use as end-of-line indicators. Do not enclose them in quotes. You can combine two characters with a plus (+) sign, for example, CHR(141) + CHR(138).

<end-of-line exp>	Represents
<i>not supplied</i>	Hard CR/LF (0D0A Hex)
CHR(141)	Soft carriage return (U.S.) (8D Hex)
CHR(255)	Soft carriage return (Europe) (FF Hex)
CHR(138)	Soft linefeed (U.S.) (8A Hex)
CHR(0)	Soft linefeed (Europe) (00 Hex)
CHR(13)	Hard carriage return (0D Hex)
CHR(10)	Hard linefeed (0A Hex)

You can't enter hexadecimal numbers directly for <end-of-line exp>, but you can combine them by adding their decimal equivalents. For example, 0D0A Hex equals CHR(13) + CHR(10); CHR(13) is 0D Hex, and CHR(10) is 0A Hex. Use HTOI() to convert a Hex number to its decimal equivalent.

Description

FGETS() reads and returns a character string from the file you specify, starting at the file pointer position and continuing up to but not including the first end-of-line character it encounters. If you don't specify a number of characters to return with *<characters expN>*, the maximum number of characters FGETS() can return is 32766. If FGETS() encounters the end-of-line character before reading the number of characters you specify with *<characters expN>*, it returns the number of characters before the end-of-line character.

FGETS() truncates the return string at the first 00 hexadecimal character, CHR(0), it encounters. If you think the file contains 00 hexadecimal characters, use FREAD() and read one byte at a time to locate them.

FGETS() returns an empty string ("") if it's unsuccessful, such as when the file pointer is at the end of the file. Use FEOF() and FERROR() to determine if the file pointer is at the end of the file or if another condition exists which would prevent FGETS() from returning a value.

If FGETS() encounters an end-of-line character, it positions the file pointer at the character after that character. Otherwise, FGETS() positions the file pointer at the character after the last character it returns. Use FSEEK() to move the file pointer before or after using FGETS().

Except for one feature, FREAD() and FGETS() are identical; FREAD() returns end-of-line characters while FGETS() does not. That is, if FGETS() encounters an end-of-line character, it stops reading, even if it hasn't yet read *<characters expN>* characters. FREAD(), on the other hand, includes end-of-line characters in the string it returns.

Example

The following example creates a text file named TEST.TXT and appends 9 lines of the string "123456789". The program then uses FGETS(), in combination with a character counter (Cnt) and a line counter (Cnt2), to output 9 progressively longer lines of text starting with 1 character and increasing to the full 9 character string. FSEEK() and Cnt2 incremented in 11 byte intervals positions the file pointer at each successive line:

```
nHandle=FCREATE("Test.TXT","RW")      && Create file
* Input text data
FOR i=1 TO 9                            && Loop for 9 lines
    FPUTS(nHandle,"123456789")          && Append text
NEXT i                                  && Increment Cnt
* Output text data
FSEEK(nHandle,0,0)                      && Moves pointer to top of file
CLEAR                                   && Clears the Command window results pane
Cnt=1
Cnt2=11                                && 11 bytes/line
DO WHILE .NOT. FEOF(nHandle)
    ? FGETS(nHandle,Cnt)                && Display string of length Cnt
    FSEEK(nHandle,Cnt2)                 && Position pointer
    Cnt=Cnt+1                           && Increment Cnt
    Cnt2=Cnt2+11                        && Increment Cnt2
ENDDO
FCLOSE(nHandle)                        && Close Test.TXT
```

F

Portability

Not supported in dBASE III PLUS.

See Also

FCREATE(), FEOF(), FERROR(), FOPEN(), FPUTS(), FREAD(), FSEEK(), HTOI()

FIELD()**Fields and records**

Returns the name of the field in a specified position of a table.

Syntax

FIELD(<field number expN> [, <alias>])

<field number expN> The position of the field whose name you want returned.

<alias> A work area number (1 through 225), letter (A through J), or alias name. The work area letter or alias name must be enclosed in quotes.

Description

FIELD() returns the name of a field in a table based on the specified *<field number expN>* parameter. Fields are numbered from 1 to 1024 based on their position in the table structure.

If you do not specify a work area, FIELD() returns the name of the field in the current table. FIELD() returns an empty string ("") if the table does not contain a field in the specified position.

Example

The following example uses FIELD() to return names of the fields in a table to generate a data entry form. This code could be used as a generic data entry procedure when the table is undetermined:

```
SET TALK OFF
CLEAR
USE Clients                && Use any table
APPEND BLANK
Fldcnt=1
DO WHILE LEN(FIELD(Fldcnt))>0
    mField=FIELD(Fldcnt)
    @ ROW()+1,0 SAY mField+;
    REPLICATE(".",25-LEN(mField)) GET &mField
    Fldcnt = Fldcnt + 1
ENDDO
READ
CLEAR
SET TALK ON
```

See Also

DBF(), FLENGTH()

FILE()

Disk and file utilities

Tests for the existence of a file. Returns .T. if the file exists and .F. if it doesn't.

Syntax

FILE(<filename expC>)

<filename expC> The name of the file that is searched for. Wildcard characters are not supported.

If you specify a file without including its path, dBASE looks for the file in the current directory, then in the path you specify with SET PATH. If you specify a file without including its extension, dBASE does not assume an extension.

F

Description

Use FILE() to determine whether a file exists.

If the file that you want to search for has an extension, you must include that extension in <filename expC>. If the file is not on the default drive, you must specify a drive designation, and if the file is not in the current directory or in the path you specify with SET PATH, you must specify the directory path.

Example

The following examples use FILE():

```
? FILE("C:\Command.com")      && Returns .T.
? FILE("Company.dbf")          && Returns .T.
? FILE("Bak\Nosuch")           && Returns .F.
```

FILE() is often used to test for the existence of a table before opening it:

```
DO WHILE .NOT. FILE("B:Company.dbf")
  ? "Insert the back up disk in drive B"
  WAIT
ENDDO
USE B:Company
```

See Also

DIR, DISPLAY FILES, GETFILE(), PUTFILE(), SET DIRECTORY, SET PATH

FIND

Table organization

Locates the first record of an indexed table whose key value matches the specified key value.

Syntax

FIND <search key literal>

<search key literal> Part or all of the index key value of a record to search for. Don't enclose <search key literal> in quotes (single or double) or square brackets.

Description

The FIND command positions the record pointer to the first record in an indexed table that matches the specified literal key value. FIND can search only for characters or a numeric or float value contained in the index key field. The match with the index key must be exact if SET EXACT is ON.

An indexed search is similar to looking up a topic in a book index and turning directly to the appropriate page. Once an index is created, FIND or SEEK can use it to quickly identify appropriate records.

FIND begins searching at the top of the index and stops when either a match is found or the end of the index is reached. If a match is found (FOUND() returns .T.), the record pointer of the associated table is positioned at the record containing the match.

Other records whose key field matches the specified characters or number can be accessed by using SKIP. SKIP advances the record pointer to the next indexed record, which includes other records with matching key fields if they exist.

Unlike using the CONTINUE command following the LOCATE command, SKIP doesn't search for a match; it moves the pointer one record whether or not you skipped to a record that matches the FIND argument.

If FIND is unsuccessful, dBASE returns the message **Find not successful** and positions the record pointer beyond the last record of the index. (EOF() is .T.; FOUND() is .F.)

Unless the key field contains leading spaces, the command-line argument need not be delimited (enclosed in single or double quotation marks or brackets). The entered expression is considered to start with the first nonblank character. However, if you are searching for a key containing leading spaces, delimiters must be used. Within the delimiters, enter the exact character string, including leading spaces. You can also store a literal character string or numeric value in a memory variable and use the memory variable to specify the search expression in the FIND command.

The search commands FIND, SEEK, and LOCATE are designed for different situations. FIND and SEEK conduct more rapid searches than LOCATE, but can be used only in limited situations. Both require an indexed file and can search only for values of the key expression. SEEK offers greater flexibility than FIND by accepting any expression.

If the information you are searching for is in an unindexed file or is not contained in the key field of an indexed file, use LOCATE. LOCATE accepts an expression of any data type as input and can search any field of a table for that value. For large tables, a sequential search using LOCATE can be slow. In such cases, use INDEX to create a new index, and then use SEEK or FIND.

Example

The following example uses FIND to locate the first occurrence of a company in Illinois:

```
USE Company EXCLUSIVE
INDEX ON State_Prov TAG State
FIND IL && Find the first Illinois record
IF FOUND()
  ? "All Companies in Illinois"
  LIST FIELDS Company, State_Prov ;
  WHILE State_Prov="IL"
```

```
ELSE
  ? "Illinois was not found"
ENDIF
```

See Also

CONTINUE, EOF(), FOUND(), LOCATE, SEEK, SEEK(), SET EXACT, SKIP

FKLABEL()

Keyboard and mouse events

F

Returns the name of a programmable function key.

Syntax

FKLABEL(<expN>)

<expN> The programmable function key whose name to return. Use FKLABEL(0) to return the value of the first programmable function key, FKLABEL(1) to return the second, etc.

Description

Use FKLABEL() to return the name assigned to a function key by the current system. You can then use the returned value with SET FUNCTION, SET KEY, or ON KEY LABEL to assign a statement or command to a function key.

Since different computer manufacturers give different names to function keys, your application may lack portability if you reference a function key by name. FKLABEL() extracts the function key's name from the current system, ensuring that the name is correct when you move your application from system to system.

On most IBM-compatible computers, programmable function keys include *F1* to *F10*, *Ctrl+F1* to *Ctrl+F10*, and *Shift+F1* to *Shift+F10*. In *Visual dBASE*, you can also program *Alt+F1* to *Alt+F10* with ON KEY or SET KEY. *Visual dBASE* doesn't support assigning values to *F11* or *F12*.

Example

The following examples use FKLABEL() in the SET FUNCTION command:

```
SET FUNCTION FKLABEL(1) TO "hello there"
SET FUNCTION FKLABEL(2) TO "DO MySub;"
* Pressing F2 is equivalent to
* typing "hello there"
* Pressing F3 will call subroutine MySub
```

Notice that:

```
? FKLABEL(0)      && returns F1
? FKLABEL(9)      && returns F10
? FKLABEL(10)     && returns CTRL+F1
? FKLABEL(19)     && returns CTRL+F10
? FKLABEL(20)     && returns SHIFT+F1
? FKLABEL(29)     && returns SHIFT+F10
```

See Also

FKMAX(), ON KEY, SET FUNCTION, SET KEY

FKMAX()**Keyboard and mouse events**

Returns the number of programmable function keys.

Syntax

FKMAX()

Description

Different computer manufacturers provide different sets of function keys, and you may need to allow for this when you transfer your application to other systems. FKMAX() detects how many function keys are available in the current system, letting you provide for alternative function key assignments when a particular system lacks certain function keys. Function key assignments are made with SET FUNCTION, SET KEY, and ON KEY LABEL.

On most IBM-compatible computers, programmable function keys include *F1* to *F10*, *Ctrl+F1* to *Ctrl+F10*, and *Shift+F1* to *Shift+F10*. In *Visual dBASE*, you can also program *Alt+F1* to *Alt+F10* with ON KEY or SET KEY. *Visual dBASE* doesn't support programming *F11* or *F12*.

Example

```
STORE FKMAX( ) TO Keycount
```

See Also

FKLABEL(), ON KEY, SET FUNCTION, SET KEY

FLDCOUNT()**Fields and records**

Returns the number of fields in a table.

Syntax

FLDCOUNT([<alias>])

<alias> A work area number (1 through 225), letter (A through J), or alias name. The work area letter or alias name must be enclosed in quotes.

Description

FLDCOUNT() returns the number of fields in the current or a specified table. If you do not specify a work area, the current work area is assumed. FLDCOUNT() returns a value of 0 if no table is open in the specified work area.

Example

The following example uses FLDCOUNT() to return the number of fields in two tables to derive a total field count for a one dimensional array that stores the field names of both tables:

```
SET TALK OFF
USE Company IN 1
Fldcnt = FLDCOUNT(1)
USE Contact IN 2
Fldcnt = Fldcnt + FLDCOUNT(2)
DECLARE Fld_Arr[Fldcnt]
x = 1
FOR Select = 1 TO 3
    FOR x_fld = 1 TO FLDCOUNT(Select)
        fld_arr[x] = FIELD(x_fld,Select)
        x = x + 1
    NEXT
NEXT
Cnt = 1
DO WHILE Cnt <= FldCnt                && Displays array contents
    ? Fld_arr[Cnt]
    Cnt=Cnt+1
ENDDO
SET TALK ON
CLOSE ALL
CLEAR ALL
```

F**See Also**

FIELD(), DISPLAY STRUCTURE, LIST STRUCTURE, RECCOUNT(), TYPE()

FLDLIST()

Fields and records

Returns the fields and calculated field expressions of a SET FIELDS TO list.

Syntax

FLDLIST([<field number expN>])

<field number expN> The position of the field or calculated field expression in a SET FIELDS TO list whose name you want returned. If you do not specify a field number, FLDLIST() returns the entire field list, up to 254 characters.

Description

FLDLIST() returns the field or calculated field expression in a SET FIELDS TO list that corresponds to a specified field number. If you do not specify a field number, FLDLIST() returns the entire field list, up to 254 characters; remaining characters are truncated. Each field name or expression in the field list is separated by a comma. FLDLIST() always returns fully-qualified field names, that is, it includes the table or alias name. For read-only fields, FLDLIST() appends "/R" to the field name.

FLDLIST() returns a value even if SET FIELDS is OFF. If the specified field number exceeds the number of items in the field list, FLDLIST() returns an empty string ("").

LENGTH()

Example

The following example demonstrates using FLDLIST() with and without an alias value:

```
USE Company IN 1
USE Contact Alias Reps IN 2
SELECT 1
SET FIELDS TO Company, City
? FLDLIST(1)                                && Returns Company->Company
SELECT 2
SET FIELDS TO CompCode, Contact
? FLDLIST(1)                                && Returns Company->Company
? FLDLIST(2)                                && Returns Company->City
? FLDLIST(3)                                && Returns Reps->CompCode
? FLDLIST()                                  && Returns full list
```

Portability

Not supported in dBASE III PLUS.

See Also

SET FIELDS

LENGTH()

Fields and records

Returns the length of the field in a specified position of a table.

Syntax

LENGTH(<field number expN> [, <alias>])

<field number expN> The position of the field whose length you want returned. Fields are numbered from 1 to 1024 according to their position in the table structure.

<alias> A work area number (1 through 225), letter (A through J), or alias name. The work area letter or alias name must be enclosed in quotes.

Description

LENGTH() returns the length of a field in a table based on the specified *<field number expN>* parameter. The field length for numeric and float fields includes decimal digits and 1 digit for the decimal point character. LENGTH() returns 1 for date and memo fields. It returns 0 if the table does not contain a field in the specified position.

If you do not specify a work area, LENGTH() returns the length of the field in the current table.

Example

The following example uses LENGTH() to return the width of the Company field (from the structure). This value is then used to compute the number of hyphens to print by subtracting the number of characters present in the field of the current record (derived from LEN(TRIM(Company))). The number 13 is then added to this amount

because of the 15 space margin between columns provided by the expression AT FLENGTH(2)+15 associated with the Contact column:

```
USE Clients
? CENTER("List of Client Companys and Contacts")
?
SCAN
? Trim(Company)+REPLICATE("-", (FLENGTH(2)-LEN(TRIM;
  (Company))+13)), Contact AT FLENGTH(2)+15
ENDSCAN
```

Portability

Not supported in dBASE IV or dBASE III PLUS.

See Also

FDECIMAL(), FIELD()

F

FLOCK()

Shared data

Locks the current table or a specified alias table, returning .T. if successful.

Syntax

FLOCK([<alias>])

<alias> A work area number (1 through 225), letter (A through J), or alias name. The work area letter or alias name must be enclosed in quotes. If you don't include <alias>, FLOCK() locks the current table.

Description

Use FLOCK() to lock the current table or an alias table file, preventing others from using the table.

When you lock a table with FLOCK(), only you can make changes to it. However, unlike USE...EXCLUSIVE and SET EXCLUSIVE ON, FLOCK() does let other users view the locked table while you are using it. When you lock a table with FLOCK(), it remains locked until you issue UNLOCK or close the table.

FLOCK() is similar to RLOCK(), except that FLOCK() locks an entire table, while RLOCK() lets you lock specific records of a table. Use FLOCK(), therefore, when you need to have sole access to an entire table or related tables—for example, when you need to update multiple tables related by a common key.

All commands that change table data cause dBASE to attempt to execute an automatic record or file lock. If dBASE fails to execute an automatic record or file lock, it returns an error. You might want to use FLOCK() for event trapping, testing for its return value rather than for an error condition.

FLOCK() can lock a shared table even if another network user is viewing data contained in the table. FLOCK() is unsuccessful only if another user has explicitly

FLOCK()

locked the table or a record in the table, or is using a command that automatically locks the table or a record in the table.

When SET REPROCESS is set to 0 (the default) and FLOCK() can't immediately lock a table, dBASE prompts you to attempt the lock again or cancel the function. Until you choose to cancel the function, FLOCK() repeatedly attempts to lock the table. Use SET REPROCESS to eliminate being prompted to cancel the FLOCK() function, or to set the number of locking attempts. If FLOCK() returns .F., you can reissue FLOCK().

When you set a relation to a parent table with SET RELATION and then lock the table with FLOCK(), dBASE attempts to lock all child tables. For more information about relating tables, see SET RELATION.

Example

This example loops until it can lock the file with FLOCK() or until the user decides to stop trying:

```
RecordWasRead=.T.
Again=.T.
SET REPROCESS TO 10
USE Company SHARED          && exclusive off
DO WHILE Again
  IF FLOCK()                  && Can dBASE lock the file?
    DO Compmod                && Yes: Update Company.DBF
    RecordWasRead=.T.
    Again=.F.
    UNLOCK                    && Allow other users to change
                              && the file.
  ELSE                        && FLOCK() returns .F.
    CLEAR
    ? "Company can't be locked."
    Wait "Try Again? " TO Ans
    IF UPPER(Ans)="Y"
      Again = .T.
    ELSE
      Again = .F.
    ENDIF
  ENDIF
ENDDO
USE
SET REPROCESS TO 0           && The default
```

Portability

Not supported in dBASE III PLUS.

See Also

BEGINTRANS(), LOCK(), RLOCK(), SET EXCLUSIVE, SET LOCK, SET RELATION, SET REPROCESS, UNLOCK, USE

FLOOR()

Numeric data

Returns the nearest integer that is less than or equal to a specified number.

Syntax

FLOOR(<expN>)

<expN> A numeric or float number, from which to determine and return the integer that is less than or equal to it.

Description

FLOOR() returns the nearest integer that is less than or equal to <expN>. If you pass a number with any digits other than zero (0) as decimal digits, FLOOR() returns the nearest integer that is less than the number. If you pass an integer to FLOOR(), or a number with only zeros for decimal digits, it returns the integer equal to the number.

For example, if the default number of decimal places is 2,

- FLOOR(2.10) returns 2.00
- FLOOR(-2.10) returns -3.00
- FLOOR(2.00) returns 2.00
- FLOOR(2) returns 2
- FLOOR(-2.00) returns -2.00

Use SET DECIMALS to set the number of decimal places FLOOR() displays.

When you pass a positive number to it, FLOOR() operates exactly like INT(). See the table in the description of INT() that compares INT(), FLOOR(), CEILING(), and ROUND().

The value returned by FLOOR() has the same data type as <expN>.

Example

The following example uses FLOOR() to return the passed value, rounded down to the next whole number:

```
SET DECIMALS TO 2
price = 129.95
tax   = .075
total = price + (price*tax)
? "The price of the upgrade is " + STR(price,6,2)
? "    The state sales tax is " + STR(tax*100,5,2) + "%"
? "    The amount due is " + STR(total,6,2)
?
? "'NO CHANGE REQUIRED' Sale Price is " + STR(FLOOR(total),6,2)
```

Another example of FLOOR() is shown in the example for INT().

Portability

Not supported in dBASE III PLUS. In dBASE IV, FLOOR() doesn't display any decimal places, regardless of the value of SET DECIMALS.

F

See Also

CEILING(), INT(), ROUND(), SET DECIMALS

FLUSH

Fields and records

Writes data buffers to disk and releases unallocated memory.

Syntax

FLUSH

Description

Use FLUSH to protect data integrity and maximize available memory.

When you open a table and its associated index and memo files, *Visual* dBASE loads a certain number of records from the file into a memory buffer, along with the portion of each open index that pertains to those records. When the buffer is full or when you close tables or indexes (with CLOSE DATABASES, USE, CLOSE INDEXES, CLOSE ALL, or SET INDEX TO), *Visual* dBASE writes the records back to disk, storing any modifications you have made. FLUSH allows you to save information from the data buffer to disk without first closing tables or indexes. FLUSH saves information in tables and associated files open in work areas other than the current work area.

Use FLUSH when you need to store critical information to disk that could otherwise be lost. However, don't use FLUSH too frequently, as it slows execution. For example, in an order-entry application in which only a few orders are entered each hour, FLUSH can save data that might be lost if the power is inadvertently turned off; since orders are entered infrequently, the time needed to execute FLUSH is not important.

Example

The following example uses FLUSH after two different data input scenarios to clear the buffers when AUTOSAVE is set to OFF:

```

USE Company IN SELECT()
SELECT Company
SET AUTOSAVE OFF
APPEND BLANK
@ 2, 2 SAY "Company Code" GET Company->CompCode
@ 3, 2 SAY "Company Name" GET Company->Company
@ 5, 8 SAY "Street" GET Company->Street1
@ 6,15 GET Company->Street2
@ 7,10 SAY "City" GET Company->City
@ 7,35 SAY "State" GET Company->State_Prov
@ 8,35 SAY "Zip" GET Company->Zip_P_Code
READ
FLUSH
CLEAR
APPEND
FLUSH
CLOSE ALL

```

Portability

Not supported in dBASE IV or dBASE III PLUS.

See Also

CLEAR, CLOSE..., MEMORY()

FNAMEMAX()**Windows 95**

Returns the maximum allowable file-name length on a given drive or volume.

F**Syntax**

FNAMEMAX([<expC>]

<expC>

The drive letter, or name of the volume, to check. If <expC> is not provided, the current drive/volume is assumed. If the drive/volume does not exist, dBASE returns an error message.

Description

FNAMEMAX() checks the drive or volume specified by <expC> and returns the maximum file-name length allowed for files on that drive/volume. This function is only useful on a system running the Windows 95 operating system. Under Windows 3.1, FNAMEMAX() always returns 12.

Example

The following example uses FNAMEMAX() to determine the maximum allowable file-name

```
length on drive C:
? FNAMEMAX("C")
255
```

Portability

Not supported in dBASE IV or dBASE III PLUS.

See Also

FSHORTNAME()

FOPEN()**Low-level access**

Opens a specified file. Returns the *file handle number* of the file if successful or -1 if unsuccessful.

Syntax

FOPEN(<filename expC> [, <access expC>])

FOPEN()

<filename expC> The name of the file to open and whose file handle number to return. By default, dBASE looks for the file in the current directory, then the directory specified in SET PATH. Specify a full path name for <filename expC> to open a file in another directory.

<access expC> The access level of the file being opened, as shown in the following table. *Write* means you can change (overwrite) data in the file, and *append* means you can add data to the end of the file. If you try to overwrite data in a file that has append access but not write access, the data is added to the end of the file.

<access expC>	Access level
not supplied	Read-only
"R"	Read-only
"W"	Write-only
"A"	Append-only
"RW" or "WR"	Read and write
"AR" or "RA"	Read and append
"AW" or "WA"	Write and append

Description

Use FOPEN() to open a file with a name you specify, assign the file the level of access you specify, and return the file handle number DOS assigns to the file. If dBASE can't open the file (for example, if the file is already open), FOPEN() returns -1.

To use other low-level functions, such as FREAD() and FWRITE(), first open a file with FOPEN() or FCREATE(). Both functions also provide you with the file handle number you need to pass to other low-level functions.

When you open a file with FOPEN(), the file pointer is positioned at the first character in the file. Use FSEEK() to position the file pointer before reading from or writing to a file.

Example

The following example uses FOPEN() to open the README file located on the default install directory and sends it to the printer by using SET PRINTER ON, a FEOF() loop and FGETS():

```
SET PATH TO C:\VISUALDB           && Parent directory
IF FILE("Readme.TXT")             && Check for file
  nHandle=FOPEN("Readme.TXT","R") && Open read only
ENDIF
IF nHandle>0                       && Did file open?
  SET PRINTER TO LPT1              && Output to print
  DO WHILE .NOT. FEOF(nHandle)
    ? FGETS(nHandle)               && Return a line
  ENDDO
  SET PRINTER TO
ELSE
  ? "Unable to open file. File error #:", LTRIM(STR(FERROR()))
ENDIF
IF nHandle=-1
```

```

? "Unable to close file. File error #:", LTRIM(STR(FERROR()))
ELSE
  IF FCLOSE(nHandle)
    ?
    ? "File closed"
  ELSE
    ? "Unable to close file. File error #:", LTRIM(STR(FERROR()))
  ENDIF
ENDIF
RETURN

```

Portability

Not supported in dBASE III PLUS.

See Also

FCLOSE(), FCREATE(), FERROR(), FILE(), GETFILE(), SET PATH

F

FOR()

Table organization

Returns the FOR clause used to create a specified index file or tag.

Syntax

FOR([[<.mdx filename expC>] <index position expN> [, <alias>]])

<.mdx filename expC> The multiple index file that contains the index tag you want to check.

<index position expN> The position of an index tag in an .MDX file or the position of an index file in the list of open indexes for the current or a specified table.

<alias> A work area number (1 through 225), letter (A through J), or alias name. The work area letter or alias name must be enclosed in quotes.

Description

FOR() returns the FOR clause used with the INDEX command to create a specified .MDX tag for the current or specified table. The FOR() function evaluates the FOR clause for the master index for the current table, the index tag in a specified position within the list of open indexes for a table, or the index tag within a specified .MDX file.

If no .MDX file name is specified, FOR() evaluates the index in a specified position within the list of all open indexes in the same work area. The FOR() function first checks .NDX and production .MDX tags, then checks index tags in other .MDX files.

FOR() returns an empty string ("") if there isn't an index tag in the specified position, the index in the specified position is an .NDX file, or the specified index tag was not created with a FOR clause. FOR() also returns an empty string ("") if you do not specify an index tag position and the table does not have a master index.

You can use TAGNO() to determine the index order number of a specified index or index tag. The order of open indexes for a specified table remains the same until you specify another index order with USE, SET INDEX, or INDEX.

Example

The following example uses FOR() to determine the conditions set in the index's FOR clause:

```
DELETE FILE Comptemp.mdx
* Create a temporary MDX file
USE Company EXCLUSIVE
INDEX ON CompCode ;
    TAG CompCode OF Comptemp FOR CompCode = "D"
INDEX ON Company ;
    TAG Company OF Comptemp
INDEX ON City ;
    TAG City OF Comptemp FOR Zip_Postal = "9"
?
? "Tag and For Clause"
?
? TAG(1),FOR(1)
* TAG(1) and FOR(1) return the TAG and FOR
* clause for the first index in the production
* MDX file, Company.mdx
? "1", TAG("Comptemp",1), FOR("Comptemp",1)
* FOR() clause of COMPCODE index is CompCode = "D"
? "2", TAG("Comptemp",2), FOR("Comptemp",2)
* FOR() clause of COMPANY index is empty
? "3", TAG("Comptemp",3), FOR("Comptemp",3)
* FOR() clause of CITY index is Zip_Postal="9"
```

Portability

Not supported in dBASE III PLUS.

See Also

INDEX, SET INDEX, SET ORDER, TAG(), TAGCOUNT(), TAGNO(), USE

FOR...NEXT

Programs

Executes the statements between FOR and NEXT the number of times indicated by the FOR statement.

Syntax

```
FOR <memvar> = <start expN> TO <end expN> [STEP <step expN>]
<statements>
[LOOP]
[EXIT]
NEXT
```

<memvar> The memory variable that's incremented or decremented and then tested each time through the loop. On a normal exit from the loop (one done without the EXIT command), the value of <memvar> is greater or less than <end expN> by the amount of <step expN>.

<start expN> The initial value of <memvar>.

<end expN> The final allowed value of *<memvar>*. If you change this value in the FOR loop, the loop doesn't recognize the change and thus continues to execute until the original value is reached.

STEP <step expN> Defines a step size (*<step expN>*) by which dBASE increments or decrements *<memvar>* each time the loop executes. By default, dBASE increments by 1. If you change this value in the FOR loop, the loop doesn't recognize the change and thus continues to increment or decrement *<start expN>* by the original *<step expN>* value.

When *<step expN>* is positive, dBASE increments *<memvar>* until it is greater than *<end expN>*. When *<step expN>* is negative, dBASE decrements *<memvar>* until it is less than *<end expN>*.

The *<step expN>* argument can be an integer, a fraction, or a floating-point number. STEP *<step expN>* must be on the same line as the FOR command.

<statements> Program lines consisting of any combination of commands, functions, user-defined functions (UDFs), and LOOP and EXIT options.

LOOP Returns program control to the top of the FOR loop and increments *<memvar>* by 1 (or increments or decrements by *<step expN>*) without executing the statements that follow LOOP and precede NEXT.

EXIT Transfers program control out of the FOR loop to the statement following NEXT without reevaluating *<memvar>* or executing the statements that follow EXIT and precede NEXT. The *<memvar>* value remains the same as when dBASE encountered the EXIT statement.

NEXT A required command that marks the end of the FOR loop. When dBASE encounters NEXT, it increments or decrements *<memvar>* according to the associated FOR statement and either returns to the associated FOR statement if the value of *<memvar>* has not reached its limit or transfers program control to the next command line.

Description

Use FOR...NEXT to execute a block of statements a specified number of times. When dBASE first encounters a FOR loop, it sets *<memvar>* to *<start expN>* and then performs the following steps:

- If *<memvar>* is in the range from *<start expN>* through *<end expN>*, dBASE executes the statements between FOR and NEXT one by one until it encounters LOOP, EXIT, or NEXT.
- If dBASE encounters LOOP or NEXT, it increments *<memvar>* by 1 (or by *<step expN>* if you used the STEP option) and returns program control to FOR.
- If *<memvar>* still evaluates to a number in the range from *<start expN>* to *<end expN>*, dBASE executes the statements in the loop again.
- If *<memvar>* evaluates to a number greater than *<end expN>* (or less than *<end expN>* if *<step expN>* is negative), the program exits the FOR loop and executes the line following NEXT.
- If dBASE encounters EXIT, it immediately transfers program control out of the loop to the statement following the associated NEXT command without incrementing or

FOUND()

decrementing *<memvar>* or executing any of the statements following EXIT and preceding NEXT.

You can nest loops and other structures, including other FOR...NEXT loops, in a FOR loop. You can nest up to 20 FOR loops in a procedure or function.

Use FOR loops to move displays in a window, process characters in a string, or access table records or array cells sequentially.

Example

The following example uses ALEN() in a FOR...NEXT loop to print out the contents of an array:

```
USE Clients
DECLARE acContact[RECCOUNT(),1]
COPY TO ARRAY acContact
USE
ShowArray(acContact)
RETURN

FUNCTION ShowArray
PARAMETER avArray
FOR i = 1 to ALEN(avArray)
    ? STR(i,3,0)+" - "+avArray[i,1]
NEXT
RETURN .T.
```

Portability

Not supported in dBASE IV or dBASE III PLUS.

See Also

DO...UNTIL, DO WHILE, IF, SCAN

FOUND()

Table organization

Indicates if the last-issued FIND, LOCATE, CONTINUE, SEEK, LOOKUP(), or SEEK() command or function finds a match.

Syntax

FOUND([*<alias>*])

<alias> A work area number (1 through 225), letter (A through J), or alias name. The work area letter or alias name must be enclosed in quotes.

Description

FOUND() returns .T. if LOCATE, CONTINUE, LOOKUP(), FIND, SEEK, or SEEK() finds a match in the current or specified table. FOUND() returns .F. if no previous search has been performed in the same work area, or if a search is unsuccessful. You can perform searches in different work areas and maintain the status of each FOUND() operation, independent of the other work areas.

If tables are linked by a SET RELATION TO command, *Visual* dBASE searches the related tables as you move in the active table with the FIND, LOCATE, SEEK, or CONTINUE commands or using the SEEK() or LOOKUP() functions. If you move the record pointer with any other command or function, FOUND() returns .F.

When SET NEAR is ON,

- FOUND() returns .T. if an exact match occurs.
- FOUND() returns .F. for a near match, and the record pointer is moved to the record whose key immediately follows the value searched for.

When SET NEAR is OFF, FOUND() returns .F. if a match does not occur.

F

Example

See the examples in SEEK, FIND, or LOCATE for an example of FOUND().

See Also

CONTINUE, EOF(), FIND, LOCATE, LOOKUP(), SEEK, SEEK(), SET NEAR, SET RELATION

FPUTS()

Low-level access

Writes a character expression and one or two end-of-line characters to a file previously opened with FCREATE() or FOPEN() and positions the file pointer after the last character written. Returns the number of characters added if successful, 0 if unsuccessful, or -1 if an error occurs.

Syntax

FPUTS(<file handle expN>, <string expC>
[, <characters expN>] [, <end-of-line exp>])

<file handle expN> The *file handle number* of the file to write the specified characters and end-of-line character to. When you open a file with FCREATE() or FOPEN(), these functions return a file handle number. Use this number as <file handle expN>. If you specify a file handle number that hasn't previously been returned by FCREATE() or FOPEN(), dBASE returns an error message.

<string expC> The character expression to write to the specified file. If you want to write only a portion of <string expC> to the file, use the <characters expN> argument.

<characters expN> The number of characters of the specified character expression <string expC> to write to the specified file, starting at the first character in <string expC>. The valid range is 0 to 32766.

<end-of-line exp> The end-of-line indicator, which can be one or two characters, to write to the specified file after the character expression. The following table lists standard

codes for use as end-of-line indicators. Do not enclose them in quotes. You can combine two characters with a plus (+) sign, for example, CHR(141) + CHR(138).

<end-of-line exp>	Represents
<i>not supplied</i>	Hard CR/LF (0D0A Hex)
CHR(141)	Soft carriage return (U.S.) (8D Hex)
CHR(255)	Soft carriage return (Europe) (FF Hex)
CHR(138)	Soft linefeed (U.S.) (8A Hex)
CHR(0)	Soft linefeed (Europe) (00 Hex)
CHR(13)	Hard carriage return (0D Hex)
CHR(10)	Hard linefeed (0A Hex)

You can't enter hexadecimal numbers directly for <end-of-line exp>, but you can combine them by adding their decimal equivalents. For example, 0D0A Hex equals CHR(13) + CHR(10); CHR(13) is 0D Hex, and CHR(10) is 0A Hex. Use HTOI() to convert a Hex number to its decimal equivalent.

Description

FPUTS() writes a character string and one or two end-of-line characters to a file. If successful, FPUTS() returns the number of bytes written to the file, including one or two for the end-of-line character or characters. If you don't have write or append access to the file, FPUTS() returns 0 and dBASE returns FERROR() number 5. If an error occurs, FPUTS() returns -1. When FPUTS() finishes executing, the file pointer is located at the character immediately after the last character written, which is the end-of-line character.

Except for one feature, FPUTS() and FWRITE() are identical; FPUTS() follows the character expression it writes with an end-of-line character while FWRITE() does not.

FPUTS() writes to the file you specify starting at the current file pointer position. Use FSEEK() to move the file pointer before or after you use FPUTS().

If the file pointer is at the end of the file, FPUTS() appends the specified string and the end-of-line character to the file. If the file pointer is not at the end of the file and you have write access to the file, FPUTS() overwrites existing text at the file pointer position. If you have append access (but not write access) to the file, FPUTS() appends text to the end of the file, regardless of the file pointer position, and dBASE does not return an error.

Example

The following example creates a text file named TEST.TXT and appends a progressively larger number of characters from the memory variable "String". The first pass through the FOR...NEXT loop appends the character "1" of "String" followed by soft carriage return and soft line feed characters. Subsequent passes append additional characters from String as i increments to 9. The program then uses FGETS in a .NOT. EOF() loop to output the contents of TEST.TXT, which will display 9 lines of text counting from 1 to 9 starting with 1 character:

```

nHandle=FCREATE("Test.TXT","RW")           && Create file
* Input text data
String="123456789"
FOR i=1 TO 9                                && Loop for 9 lines
    FPUTS(nHandle,String,i,CHR(13)+CHR(10))
* Append i number of characters from String
* followed by a CR/LF.
NEXT i                                       && increments i
* Output text data
FSEEK(nHandle,0,0)                          && Moves pointer to top of file
CLEAR                                       && Clear the Command window results pane
DO WHILE .NOT. FEOF(nHandle)
    ? FGETS(nHandle)                       && Display line
ENDDO
FCLOSE(nHandle)                            && Close Test.TXT

```

F

Portability

Not supported in dBASE III PLUS. In dBASE IV, FPUTS() fails and dBASE returns an error when used with a file that doesn't have write or append access.

See Also

FCREATE(), FEOF(), FERROR(), FGETS(), FOPEN(), FSEEK(), FWRITE(), HTOI(), SUBSTR()

FREAD()

Low-level access

Returns a specified number of characters from a file previously opened with FCREATE() or FOPEN() and positions the file pointer after the last character returned.

Syntax

FREAD(<file handle expN>, <characters expN>)

<file handle expN> The *file handle number* of the file whose characters to read and return. When you open a file with FCREATE() or FOPEN(), these functions returns a file handle number. Use this number as <file handle expN>. If you specify a file handle number that hasn't previously been returned by FCREATE() or FOPEN(), dBASE returns an error message.

<characters expN> The number of characters to return from the specified file. Acceptable values are 0 to 32766; if <characters expN> is less than 0 or greater than 32766, dBASE uses 0 or 32766, respectively.

Description

FREAD() returns the number of characters you specify from the file you specify. FREAD() starts reading and storing characters from the current file pointer position and repositions the file pointer to the character immediately after the last character read. Use FSEEK() to move the file pointer before or after you use FREAD().

Except for one feature, FREAD() and FGETS() are identical; FREAD() returns end-of-line characters while FGETS() does not. For more information, see FGETS().

FSEEK()

Example

The following example uses FREAD() to display 600 characters at a time from the README.TXT file. The WAIT command within the loop allows the user to review the file one page at a time.

```
SET PATH TO C:\VISUALDB                && Parent directory
nHandle=FOPEN("Readme.TXT", "R")      && Open read only
DO WHILE .NOT. FEOF(nHandle)
    ? FREAD(nHandle,600)                && Display next 600 characters
    ?                                   && Spacing above Wait
    WAIT "Browse at your leisure-;
        Press any key to continue"
    ?                                   && Spacing below Wait
ENDDO
RETURN
```

Portability

Not supported in dBASE III PLUS.

See Also

FCREATE(), FEOF(), FERROR(), FGETS(), FOPEN(), FSEEK(), FWRITE()

FSEEK()

Low-level access

Moves the file pointer a specified number of bytes in a file previously opened with FCREATE() or FOPEN(), and returns the number of bytes from the beginning of the file to the file pointer.

Syntax

FSEEK(<file handle expN>, <bytes expN> [, <position expN>])

<file handle expN> The *file handle number* of the file in which to move the file pointer.

When you open a file with FCREATE() or FOPEN(), these functions return a file handle number. Use this number as <file handle expN>. If you specify a file handle number that hasn't previously been returned by FCREATE() or FOPEN(), dBASE returns an error message.

<bytes expN> The number of bytes to move the file pointer in the specified file. If <bytes expN> is negative, the file pointer moves toward the beginning of the file. If <bytes expN> is 0, the file pointer moves to the position you specify with <position expN>. If <bytes expN> is positive, the file pointer moves toward the end of the file or beyond the end of the file.

<position expN> The number 0, 1, or 2, indicating a position relative to the beginning of the file (0), to the file pointer's current position (1), or to the end of the file (2). The default is 0.

Description

FSEEK() moves the file pointer in the file you specify, and returns the number of bytes from the beginning of the file to the file pointer's new position. If an error occurs, FSEEK() returns -1.

The movement of the file pointer is relative to the beginning of the file unless you specify otherwise with *<position expN>*. For example, FSEEK(filenum, 5) moves the file pointer five characters from the beginning of the file while FSEEK(filenum, five, 1) moves it five characters forward from its current position. You can move the file pointer beyond the end of the file, but you can't move it before the beginning of the file.

To move the file pointer to the beginning of a file, use FSEEK(*<file handle expN>*, 0). To move it to the end of a file, use FSEEK(*<file handle expN>*, 0, 2).

FGETS(), FPUTS(), FREAD(), and FWRITE() also move the file pointer as they read from or write to the file.

Example

See FGETS() for an example of using FSEEK().

Portability

Not supported in dBASE III PLUS.

See Also

FCREATE(), FEOF(), FOPEN(), FREAD(), FWRITE()

F

FSHORTNAME()

Windows 95

Returns the short name (i.e. the DOS compatible name) of a file created under Windows 95.

Syntax

FSHORTNAME(*<filename expC>*)

<filename expC> The name of the file to check. If the file is not in the current directory, you must provide the path. If the file does not exist, dBASE returns an error message.

Description

FSHORTNAME() checks the file specified by *<filename>* and returns a name for the file following the DOS file naming convention (eight character file name, three character extension). If SET FULLPATH is ON, the path is also returned. This function is only useful on a system running the Windows 95 operating system. Under Windows 3.1, FSHORTNAME() returns the file-name.

Example

The following example uses FSHORTNAME() to check the short name of a table:

FSIZE()

```
? FSHORTNAME("ANIMAL_LISTINGS_TABLE.DBF")  
ANIMAL~1.DBF
```

Portability

Not supported in dBASE IV or dBASE III PLUS.

See Also

FNAMEMAX()

FSIZE()

Disk and file utilities

Returns the size of a file in bytes.

Syntax

FSIZE(<filename expC>)

<filename expC> The name of the file to evaluate. Wildcard characters are not supported.

If you specify a file without including its path, dBASE looks for the file in the current directory, then in the path you specify with SET PATH. If you specify a file without including its extension, dBASE does not assume an extension.

Description

Use FSIZE() to determine the size of a file on disk.

When you update a file, dBASE changes the file's size, which appears next to the file name in a directory listing. For example, when the user edits a table, dBASE changes the size on the table file when the file is closed. FSIZE() reads the size and returns its current value.

If the file that you want to evaluate has an extension, you must include that extension in <filename expC>. If the file is not on the default drive, you must specify a drive designation, and if the file is not in the current directory or in the path you specify with SET PATH, you must specify the directory path.

If dBASE cannot find the file, it returns an error. Therefore, you may want to test for its existence with FILE() before issuing FSIZE(). FLUSH does not update a file's size.

If <filename expC> is present in the current directory and also exists in the SET PATH directory, FSIZE(<filename expC>) without path information returns information on the file in the current directory.

Example

The following example finds the size of the Company table and, if there is enough space on drive B, writes a copy to that drive:

```
Needed1=FSIZE("Company.dbf")  
Needed2=FSIZE("Company.dbt")      && the memo file  
Needed=Needed1+Needed2  
Available=DISKSPACE(2)
```

```

IF Needed <= Available
  RUN COPY Company.dbf B:
  RUN COPY Company.dbt B:
ELSE
  ? "Available", Available
  ? "Needed  ", Needed
  WAIT "Warning. Not enough space on drive B:"
ENDIF

```

Portability

Not supported in dBASE III PLUS.

F

See Also

FDATE(), FILE(), FLUSH, FTIME(), SET DIRECTORY, SET PATH

FTIME()

Disk and file utilities

Returns the time stamp for the file named *<filename expC>*.

Syntax

FTIME(*<filename expC>*)

<filename expC> The name of the file to evaluate. Wildcard characters are not supported.

If you specify a file without including its path, dBASE looks for the file in the current directory, then in the path you specify with SET PATH. If you specify a file without including its extension, dBASE does not assume an extension.

Description

Use FTIME() to determine the time of day when the last change was made to a file on disk.

When you update a file, dBASE changes the file's time stamp to the current operating system time when the file is written to disk. For example, when the user edits a table, dBASE changes the time stamp on the table file when the file is closed. FTIME() reads the time stamp and returns its current value.

If the file that you want to evaluate has an extension, you must include that extension in *<filename expC>*. If the file is not on the default drive you must specify a drive designation, and if the file is not in the current directory or in the path you specify with SET PATH, you must specify the directory path.

If dBASE cannot find the file, it returns an error. Therefore, you may want to test for its existence with FILE() before issuing FTIME(). FLUSH does not update a file's time stamp.

If *<filename expC>* is present in the current directory and also exists in the SET PATH directory, FTIME(*<filename expC>*) without path information returns information on the file in the current directory.

Example

This example compares the date and time stamps on COMPANY.DBF and its backup on the B drive. If the backup is the same day as the current table but an earlier time then a backup is recommended:

```
DO CASE
CASE FDATE("B:Company.dbf") = FDATE("Company.dbf") .AND.;
    FTIME("B:Company.dbf") < FTIME("Company.dbf")
    * Same day but different times
    ? "Backup recommended"
    WAIT
CASE FDATE("B:Company.dbf") < FDATE("Company.dbf")
    * Backup is earlier date than current table
    Difference= FDATE("Company.dbf") - FDATE("B:Company.dbf")
    ? Difference, " days since last backup"
    ? "Backup recommended"
    WAIT
ENDCASE
```

Portability

Not supported in dBASE III PLUS.

See Also

FILE(), FLUSH, FSIZE(), FDATE(), SET DIRECTORY, SET PATH

FUNCTION

Programs

Defines a user-defined function (UDF) in a program file. This command is supported primarily for compatibility with dBASE IV, in which there were significant syntactic and functional differences between UDFs and procedures (defined with the PROCEDURE command). In *Visual* dBASE, procedures and UDFs use the same syntax, and can be used and called in the same ways. Therefore, making a distinction between procedures and functions is no longer necessary, and using PROCEDURE is recommended in *Visual* dBASE. For more information, see PROCEDURE.

Portability

In dBASE IV, you could have a procedure and a UDF with the same name available at the same time, because they were called differently. In *Visual* dBASE, if a procedure and a UDF of the same name are available, the first one declared is the only one recognized. For more portability information, see PROCEDURE.

FUNIQUE()

Disk and file utilities

Creates a unique file name.

Syntax

FUNIQUE(<expC>)

<expC> A file-name skeleton, which can include wildcards and any valid file-name characters.

Description

Use FUNIQUE() to create a unique file name with random numbers and letters you specify. For example, use FUNIQUE() to create temporary files without overwriting existing files.

To specify a file name of a specific length, or to specify which characters in the file name should be random numbers, use the ? wildcard character.

FUNIQUE() generates the new file name by replacing each wildcard character with a random number, then looking in the current or specified directory for a file name that matches the new file name. If no match is found, FUNIQUE() creates the file name and returns the name. If a match is found, FUNIQUE() tries again until a unique file name is found. If no combination of random numbers is successful, FUNIQUE() returns an empty string.

If you omit <expC>, FUNIQUE() creates an 8-character file name with no extension, composed entirely of random numbers.

Example

The following example uses FUNIQUE() to obtain a unique file name for a temporary table to which totals are calculated. The temporary table is then removed:

```
TempTable=FUNIQUE("Temp????.dbf")
* Using FUNIQUE, Temp???? returns Temp followed by 4 numbers and .dbf,
* for example, Temp7990.dbf, Temp8832.dbf

USE Orders EXCLUSIVE
INDEX ON Customer_n TAG Customer_n
TOTAL ON Customer_n TO &TempTable
* TempTable now has the totals for orders for each customer

USE &TempTable
BROWSE FIELDS Customer_n, Total_inv, Amt_paid
USE

ERASE &TempTable
```

Portability

Not supported in dBASE IV or dBASE III PLUS.

See Also

FCREATE(), FERROR(), FILE(), FOPEN()

F

Returns a float that is the future value of an investment.

Syntax

FV(<payment expN>, <interest expN>, <term expN>)

<payment expN> The amount of the periodic payment. Specify the payment in the same time increment as the interest and term. The payment can be negative or positive.

<interest expN> The interest rate per period expressed as a positive decimal number. Specify the interest rate in the same time increment as the payment and term.

<term expN> The number of payments. Specify the term in the same time increment as the payment and interest.

Description

Use FV() to calculate the amount realized (future value) after equal periodic payments (deposits) at a fixed interest rate. FV() returns a float representing the total of the payments plus the interest generated and compounded.

Express the interest rate as a decimal. For example, if the annual interest rate is 9.5%, <interest expN> is .095 (9.5/100) for payments made annually.

Express <payment expN>, <interest expN>, and <term expN> in the same time increment. For example, if the payment is monthly, express the interest rate per month, and the number of payments in months. You would express an annual interest rate of 9.5%, for example, as .095/12, which is 9.5/100 divided by 12 months.

The formula dBASE uses to calculate FV() is as follows:

$$fv = pmt * \frac{1 + int^{term} - 1}{int}$$

where int = rate / 100

For the future value an investment of \$350 made monthly for five years, earning 9% interest, the formula expressed as a dBASE expression looks like this:

```
? FV(350, .09/12, 60)           && Returns 26398.45
? 350*((1+.09/12)^60-1)/(.09/12) && Returns 26398.45
```

In other words, if you invest \$350/month for the next five years into an account that pays an annual interest rate of 9%, at the end of five years you will have \$26398.45.

Use SET DECIMALS to set the number of decimal places FV() displays.

Example

The following example uses FV() to calculate the future value of monthly payments made over a set period at a known interest rate:

```
LOCAL f
f = NEW GFORM()
f.Open()
```

```

CLASS GFORM OF FORM
    this.Left =          49.00
    this.Height =        15.00
    this.ColorNormal = "BG+/BG"
    this.Width =          50.00
    this.Text = "Future Value"
    this.HelpId = " "
    this.HelpFile = " "
    this.Top =            5.82

    DEFINE TEXT TXT1 OF THIS;
        PROPERTY;
            Left          5.00;;
            Height        1.00;;
            ColorNormal "BG+/BG",;
            Width         26.00;;
            Border .F.,;
            Text "Monthly installment?",;
            Top           3.00

    DEFINE ENTRYFIELD AMNT OF THIS;
        PROPERTY;
            Left          30.00;;
            Height        1.00;;
            Width         5.00;;
            Value         0,;
            Picture "9999",;
            Border .T.,;
            Top           3.00

    DEFINE TEXT TXT2 OF THIS;
        PROPERTY;
            Left          5.00;;
            Height        1.00;;
            ColorNormal "BG+/BG",;
            Width         26.00;;
            Border .F.,;
            Text "Interest rate expected?",;
            Top           5.00

    DEFINE ENTRYFIELD INT OF THIS;
        PROPERTY;
            Left          33.00;;
            Height        1.00;;
            Width         5.00;;
            Value         0.00;;
            Picture "99.99",;
            Border .T.,;
            Top           5.00

    DEFINE TEXT TXT3 OF THIS;
        PROPERTY;
            Left          5.00;;
            Height        1.00;;
            ColorNormal "BG+/BG",;
            Width         34.00;;
            Border .F.,;

```

FWRITE()

```
Text "How many monthly payments?";
Top 7.00

DEFINE ENTRYFIELD PYMTS OF THIS;
PROPERTY;
Left 40.00;;
Height 1.00;;
Width 4.00;;
Value 0;;
Picture "999",;
Border .T.,;
Top 7.00

DEFINE PUSHBUTTON RESULTS OF THIS;
PROPERTY;
Left 14.00;;
Height 2.00;;
ColorNormal "N/W",;
Width 19.00;;
OnClick {;myResult="Future Value will be: $" +
LTRIM(STR(FV(Form.Amt.Value,
(Form.Int.Value/100)/12,Form.Pymts.Value),13,2)) ;
Form.FV.Text=myResult};;
Text "Future Value",;
Default .T.,;
Top 11.00

DEFINE TEXT FV OF THIS;
PROPERTY;
Left 5.00;;
Height 1.00;;
ColorNormal "BG+/BG",;
Width 33.00;;
Border .F.,;
Text "Future Value: ",;
Top 9.00

ENDCLASS
```

Portability

Not supported in dBASE III PLUS.

See Also

PAYMENT(), PV(), SET DECIMALS

FWRITE()

Low-level access

Writes a character expression to a specified file and positions the file pointer after the last character written. Returns the number of characters added if successful, 0 if unsuccessful, or -1 if an error occurs.

Syntax

FWRITE(*<file handle expN>*, *<string expC>* [, *<characters expN>*])

<file handle expN> The *file handle number* of the file to write the specified characters and end-of-line character to. When you open a file with **FCREATE**() or **FOPEN**(), these functions returns a file handle number. Use this number as *<file handle expN>*. If you specify a file handle number that hasn't previously been returned by **FCREATE**() or **FOPEN**(), dBASE returns an error message.

<string expC> The character expression to write to the specified file. If you want to write only a portion of *<string expC>* to the file, use the *<characters expN>* argument.

<characters expN> The number of characters of the specified character expression *<string expC>* to write to the specified file, starting at the first character in *<string expC>*.

F**Description**

FWRITE() writes a character string to a file. Except for one feature, **FWRITE**() and **FPUTS**() are identical; **FPUTS**() follows the character expression it writes with an end-of-line character while **FWRITE**() does not. For more information, see **FPUTS**().

Example

The following example uses **FWRITE**() within a **SCAN** loop to write the contents of the Name field of each record in the Animals table to a text file. The program then uses **FGETS**() within a **DO WHILE .NOT. FEOF**() loop to output the contents of the text file to the Command window.

```
nFile="Animals.TXT"
nHandle=FCREATE(nFile,"RW")           && Create text file
SET PATH TO C:\VISUALDB\SAMPLES        && Make Samples directory available
USE ANIMALS
* input data
SCAN
    FWRITE(nHandle,Trim(Name)+CHR(13)+CHR(10))
* Append Name field contents plus CR/LF to text file.
ENDSCAN* output data
FSEEK(nHandle,0,0)                     && File pointer to top of file
CLEAR                                  && Clears Command window results pane
DO WHILE .NOT. FEOF(nHandle)
    ? FGETS(nHandle)                   && Display line
ENDDO
FCLOSE(nHandle)                        && Close ANIMALS.TXT
```

Portability

Not supported in dBASE III PLUS.

See Also

FCREATE(), **FEOF**(), **FERROR**(), **FOPEN**(), **FPUTS**(), **FREAD**(), **FSEEK**()

GENERATE

Error handling and debugging

Adds random records to the current table.

Syntax

GENERATE [*<expN>*]

<expN> A number of random-data records to add to the current table. The *<expN>* argument must be between 1 and 1,000,000,000 inclusive and can't result in the generation of more than 2,000,000,000 bytes (2,000MB), the maximum size of a dBASE table. If you specify a *<expN>* value that is less than or equal to 0, dBASE doesn't generate any records. If you don't specify a value for *<expN>*, dBASE prompts you for a number and waits for input.

Description

GENERATE fills a table with sample data so you can thoroughly test and debug a program. If a table contains existing records, GENERATE leaves them intact and adds *<expN>* records to the table.

When you use the GENERATE command with a table containing a memo field, dBASE creates a memo field for each record but doesn't fill its associated memo file with data.

Example

The following example uses GENERATE to add 10 new records to a temporary table created from CLIENTS.DBF.

```
USE Clients
COUNT      && Returns number of records
COPY TO Temp
USE Temp
GENERATE 10
COUNT      && Returns previous number +10
BROWSE      && Note the addition of 10 random records
```

Portability

Not supported in dBASE IV or dBASE III PLUS.

See Also

APPEND, BROWSE, CHANGE, DEBUG, DISPLAY COVERAGE, EDIT, SET COVERAGE

GETCOLOR()

Colors and fonts

Calls a dialog box in which you can define a custom color or select a color from the color palette. Returns a character string containing the red, green, and blue values for the color selected.

Syntax

GETCOLOR([<*title expC*>])

<*title expC*> A character string up to 78 characters long that appears as the title of the dialog box.

Description

Use GETCOLOR() to open a dialog box in which you can choose a color from a palette of predefined colors or create a customized color. In this dialog box, you choose and create colors in the same way you do if you use the Color Palette available when you choose Color in the Windows Control Panel.

GETCOLOR() returns a string in the format "red value, green value, blue value", as shown in the following example.

mRed = GETCOLOR()	&& choose a pure red color
? mRed	&& returns "255,0,0"
mBlue = GETCOLOR()	&& create a light blue color
? mBlue	&& returns "164,200,240"

You can use the string returned by GETCOLOR() in a related command, DEFINE COLOR, to use a specific color in a program.

Portability

Not supported in dBASE IV or dBASE III PLUS.

See Also

DEFINE COLOR

G

GETDIRECTORY()

Disk and file utilities

Displays a dialog box from which you can select a directory for use with subsequent commands.

Syntax

GETDIRECTORY([<*directory expC*>])

<*directory expC*> The initial directory to appear in the dialog box. If <*directory expC*> is omitted, the current directory appears as the initial directory.

Description

Use GETDIRECTORY() to return a directory name for use in subsequent commands, such as FCREATE(), SET DIRECTORY, or SET PATH.

GETDIRECTORY() does not return a final backslash at the end of the directory name it returns. If you want to create a full path name by concatenating a file name onto the value GETDIRECTORY() returns, include the backslash in your directory name in one of the following ways:

GETENV()

```
* Add backslash to directory returned
mCpath = GETDIRECTORY() + "\"
* Add backslash when concatenating to directory
mCfilename = "abc.txt"
mCfullname = GETDIRECTORY() + "\" + mCfilename
```

Example

The following shows three examples of GETDIR():

```
Newdir=GETDIR()
* open the directory dialog box
* If user chooses Cancel then Newdir is empty
Newdir=GETDIR("D:\Examples")
USE Company
Fullname=GETDIR()+"\"+DBF()
```

Portability

Not supported in dBASE IV or dBASE III PLUS.

See Also

FCREATE(), GETFILE(), SET DIRECTORY, SET PATH

GETENV()

Disk and file utilities

Returns the value of a DOS environment variable.

Syntax

GETENV(<expC>)

<expC> The name of the DOS environment variable to evaluate.

Description

Use GETENV() to return the current value of a DOS environment variable.

DOS environment variables are similar to dBASE memory variables. These variables are usually created with DOS commands like SET and PATH. For example, SET NAME=HARRY creates an environment variable NAME containing the character string "HARRY", and PATH=C:\DOS changes the DOS variable PATH or creates a new one.

If dBASE can't find the environment variable specified by <expC>, it returns a null string.

Example

The following examples use GETENV():

```
? GETENV("comspec") && where is command.com
? GETENV("Path")    && the current path
? GETENV("Prompt")  && the prompt
```


See Also

OS(), SET PATH

GETEXPR()**Expressions and type conversion**

Displays a dialog box in which you can create or edit an expression, and returns the expression you specify.

Syntax

```
GETEXPR([<expression expC> [, <title expC> [, <data type expC>]])
```

<expression expC> A character expression to edit. If you specify *<expression expC>*, a dialog box opens with *<expression expC>* in the Expression field. Without *<expression expC>*, the dialog box opens with nothing in the Expression field.

<title expC> A character string that appears as the title of the dialog box. Without *<title expC>*, the default title of the dialog box displays. If you want to specify a value for *<title expC>*, you must also specify a value or empty string ("") for *<expression expC>*.

<data type expC> A single character specifying the data type of the result of the expression you specify in the dialog box. The data type appears as the Result type in the dialog box. If you want to specify a value for *<data type expC>*, you must also specify a value or empty string for *<expression expC>* and *<title expC>*.

Use the following characters in *<data type expC>* for the corresponding data types:

- C for character
- D for date
- L for logical
- N for numeric
- X for any

Description

GETEXPR() displays the dBASE Expression Builder tool. Use it to build valid dBASE expressions to insert into the Text Editor, the Command window, and certain dialog box fields. For example, a user could use the Expression Builder to create a condition you then apply to a table with a SET FILTER statement.

The Expression Builder only builds expressions; it doesn't assign them. You can issue GETEXPR() in a program or in the Command window, and you can assign a variable to it so that the variable gets the value of what GETEXPR() returns.

In addition to using GETEXPR(), you can also call the Expression Builder in the following ways:

- Choose Edit | Build Expression from the menu
- Press *Ctrl+E*

G

GETFILE()

Example

The following example uses GETEXPR() to branch to the Edit an Expression dialog box for entry of a state to search for:

```
USE Clients
cState      = SPACE( LEN(Clients->State_Prov) )
cExpression = "UPPER(State_Prov)"
cCaption    = " Magic!! Build an Expression "
cType       = "L"
cKey        = ""
CLEAR
@ 1,1 SAY "Enter a state's name: " ;
      GET cState                  ;
      PICTURE "@!"

READ
IF .NOT. ISBLANK( cState )
  cExpression = cExpression + " = " + "'" + cState;
  + "'"
  cKey = GETEXPR(cExpression,cCaption,cType)
  LOCATE FOR &cKey.
  IF FOUND()
    ? Company, City
  ELSE
    CLEAR
    ? "There are no entries for " + cState
  ENDIF
ENDIF
ENDIF
USE
```

Portability

Not supported in dBASE IV or dBASE III PLUS.

See Also

SET FILTER

GETFILE()

Disk and file utilities

Displays a dialog box, from which the user can choose or enter an existing file name, and returns the file name. Returns an empty string if the user exits the dialog box by any method except by double-clicking on a file name or choosing OK.

Syntax

```
GETFILE([<filename skeleton expC>
[, <title expC>
[, <filetype expL>
[, <change filetype expL>]]]])
```

<filename skeleton expC> A character string that matches selected file names with the wildcard characters ? and *. The GETFILE() dialog box lists only those file names in the current directory that match the file name skeleton. Without *<filename skeleton expC>*, the dialog box lists all file names.

<title expC> A title displayed in the top of the dialog box. <title expC> can be up to 60 characters long. Without <title expC>, the GETFILE() dialog box displays the default title. If you want to specify a value for <title expC>, you must also specify a value or empty string ("") for <filename skeleton>.

<filetype expL> A logical value that determines whether the dialog box opens with a list of all file types (.T.) or with a list of tables in a database (.F.). The default is .T. If you want to specify a value for <filetype expL>, you must also specify a value or empty string ("") for <filename skeleton> and <title expC>.

<change filetype expL> A logical value that determines whether the user can switch between database tables and all file types while in the dialog box (.T.) or cannot switch between file types (.F.). The default is .F. If you want to specify a value for <change filetype expL>, you must also specify a value or empty string ("") for <filename skeleton> and for <title expC>, and you must specify a value for <filetype expL>.

G

Description

Use GETFILE() to retrieve a file name of your choice from a dialog box. Once the file name is retrieved, you can use it in other commands and function calls. For example, you can use GETFILE() to return a table file name so you can open a table with USE, or return the name of a text file for an editor object you create with DEFINE EDITOR. GETFILE() does not open any files.

The GETFILE() dialog box includes names of files whether they are currently open or closed. dBASE returns the full path name of the file whether SET FULLPATH is ON or OFF.

By default, the dialog box opened with GETFILE() displays file names from the current directory the first time you issue GETFILE(). After the first time you use GETFILE() and exit successfully, the subdirectory you choose becomes the default the next time you use GETFILE().

Example

The following examples use GETFILE():

F1=GETFILE()	&& Simply opens the dialog box
F2=GETFILE("*.prg")	&& Displays only program files
F3=GETFILE("*.dbf","Choose any table")	&& Selects tables and displays;
	&& title "Choose any table"
F4=GETFILE("*.dbf","Only a table",.f.)	&& Only allow user to choose tables
? "F1",F1	
? "F2",F2	
? "F3",F3	
? "F4",F4	

Portability

Not supported in dBASE IV or dBASE III PLUS.

See Also

FILE(), PUTFILE(), SET FULLPATH

GETFONT()

Colors and fonts

Calls a dialog box in which you select a character font. Returns a string containing the font name, point size, font style (if you choose a style other than Regular), and family.

Syntax

GETFONT([<*title expC*>])

<*title expC*> A character string up to 78 characters long that appears as the title of the dialog box.

Description

Use GETFONT() to place the values associated with a specified font into a character string, as shown in the following examples. If you want to add a font to the [Fonts] section of DBASEWIN.INI but don't know its exact name or family, use GETFONT(). Then add the information GETFONT() returns into DBASEWIN.INI.

mNormal = GETFONT()	&& choose Arial, Regular, 10-pt
? mNormal	&& returns "Arial,10,Swiss"
mBold = GETFONT()	&& choose Helvetica bold, 12-pt
? mBold	&& returns "Helvetica,12,B,Swiss"

Portability

Not supported in dBASE IV or dBASE III PLUS.

See Also

?

GO

Fields and records

Moves the record pointer to the specified position in a table. For Paradox and SQL tables, you can move to a specific record using bookmarks.

Syntax

GO[TO]
 BOTTOM | TOP | <*bookmark*> | [RECORD] <*expN*>
 [IN <*alias*>]

TO Include for readability only; TO has no affect on the operation of the command.

BOTTOM | TOP | <bookmark> | RECORD <expN> Specifies where to move the record pointer. The following table describes each of the available keywords or options.

Option	Description
BOTTOM	If the specified table has no master index, moves the record pointer to the last record of the table. If the table has a master index, moves the record pointer to the last record of the index.
TOP	If the specified table has no master index, moves the record pointer to the first record of the table. If the table has a master index, moves the record pointer to the first record of the index.
<bookmark>	A marker for a specific row (similar to a record pointer) for tables that don't support record numbers.
RECORD <expN>	The record number to move the record pointer to. Entering a number in the Command window is equivalent to GO <expN>. The RECORD keyword is included for readability only; it has no affect on the operation of the command.

G

IN <alias> Specifies an open table other than the current one, in which to move the record pointer. You can specify a work area number (1 through 225), letter (A through J), or alias name. The work area letter or alias name must be enclosed in quotes.

Description

GO positions the record pointer in the current or specified table or index file. GO <expN> or GO RECORD <expN> moves the record pointer to a specific record, regardless of whether a master index is open or where that record number occurs in an indexed order.

For tables that do not support record numbers (that is, Paradox and SQL tables), GO <expN> returns an error. For these type of tables, you specify a <bookmark> in place of a record pointer value. Bookmarks are a special data type that cannot be displayed or printed directly. You can return a bookmark of a specific record, for example, using the RECNO() or BOOKMARK() function. Optionally, you can store the bookmark in a memory variable and substitute that variable in the <bookmark> argument provided with the GO command.

If an index isn't in use, TOP and BOTTOM refer to the first and last records in a table. If an index file is in use for a table, TOP and BOTTOM refer to the first and last records in the index file.

With SET DELETED ON, you can move the record pointer to a record that is marked for deletion by directly specifying its 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 tables, moving the record pointer in the parent table with GOTO repositions the record pointer in a child table to a related record. If there is no related record, the child table record pointer is positioned at the end of the file. Moving the record pointer in a child table, however, doesn't reposition the record pointer in the parent table.

Example

The following example uses GO to move the record pointer in an open table:

```

SET TALK OFF
USE Clients EXCLUSIVE
INDEX ON Client_ID TAG Client_ID
SEEK "A5577"
? RECNO(), Client_ID, Company, Zip_P_Code
rec_num = RECNO()
GO TOP
? RECNO(), Client_ID, Company, Zip_P_Code
GO BOTTOM
? RECNO(), Client_ID, Company, Zip_P_Code
rec_mark = BOOKMARK()
GO rec_num
? RECNO(), Client_ID, Company, Zip_P_Code
GO rec_mark
? RECNO(), Client_ID, Company, Zip_P_Code
CLOSE ALL

```

See Also

EOF(), RECNO(), SELECT, SET DELETED, SET FILTER, SET RELATION, SKIP

HELP**Windows programming**

Activates the dBASE Help system.

Syntax

HELP

[<help topic>]

<help topic> The Help topic you access with HELP.

Description

Use the HELP command to provide information on dBASE.

The <help topic> option is a character string consisting of a single letter or a group of letters. dBASE locates the first Help topic beginning with this string and opens the Search dialog box with the topic highlighted.

Pressing *F1* executes the HELP command automatically unless you reprogram *F1* with ON KEY LABEL or SET FUNCTION.

For information on creating a customized Help system, see Chapter 14 in the *Programmer's Guide*.

Example

The command HELP, used in a program or in the Command window, can provide explanations of individual commands, features, or a user overview of Views and Tools, the Debugger or language. To access information on Views and Tools, issue the following command:

```
HELP Views and Tools
```

To access Help information on a specific command type (for example)

```
HELP CLASS
```

To access information on a feature type (for example)

```
HELP SPEEDBARS
```

See Also

HelpFile, HelpID, SET HELP TO, SET TOPIC

HOME()

Disk and file utilities

Returns the path to the directory where the DBASEWIN.EXE in use is located.

H

Syntax

```
HOME( )
```

Description

Use HOME() to identify the directory in which the currently running copy of DBASEWIN.EXE is located. When you install dBASE, the installation program (by default) installs DBASEWIN.EXE in the DOS directory \VISUALDB\BIN. HOME() returns this directory. HOME() returns the full path name whether SET FULLPATH is ON or OFF.

To identify the dBASE home directory, use _dbwinhome.

Example

HOME() returns the dBASE directory from which DBASEWIN.EXE was launched:

```
? HOME()      && F:\VISUALDB\BIN\
* This is different from _dbwinhome
? _dbwinhome  && \VISUALDB\
```

Portability

Not supported in dBASE III PLUS.

See Also

_dbwinhome, CD, MKDIR, SET DIRECTORY, SET FULLPATH, SET PATH

HTOI()

Expressions and type conversion

Returns the decimal-number equivalent of a specified hexadecimal number.

Syntax

```
HTOI(<expC>)
```

<expC> The hexadecimal number whose decimal-number equivalent to return.

Description

Use HTOI() to convert a hexadecimal number to a decimal number of float type. HTOI() is the inverse function of ITOH(), which accepts numeric and float numbers and returns a hexadecimal equivalent as a character string. Use both functions with Windows Application Programming Interface (API) calls that require hexadecimal values.

Example

The following examples use HTOI() to return the numeric value of the passed hexadecimal values:

```
? HTOI("FAFA")           && Returns 64250.0000
? HTOI("40FA")           && Returns 16634.0000
? ITOH(12345,8)          && Returns 00003039
? HTOI("    3013")       && Returns 12307.0000
```

Portability

Not supported in dBASE IV or dBASE III PLUS.

See Also

ITOH()

ID()**Shared data**

Returns the name of the current user on a *local area network* (LAN) or other multiuser system.

Syntax

ID()

Description

ID() accepts no arguments and returns the name of the current user as a character string. ID() returns an empty string when you call it on a single-user system or when a user name isn't registered on a multiuser system.

Example

The following example keeps track of the last network user to update a record. It updates a network database and puts ID(), the current user, into a field called USER:

```
PROCEDURE OkToChange && Updates a network database and logs user name
IF ID() <> ""
    REPLACE NAME WITH cName, USER with ID()
ELSE
    CLEAR
    ? "You have lost your network connection. Data not saved."
    WAIT
ENDIF
RETURN
```


Portability

Not supported in dBASE III PLUS.

See Also

CONVERT, LKSYS(), NETWORK()

IF**Programs**

Conditionally processes statements by evaluating one or more conditions and executing the statements following the first condition that evaluates to true.

Syntax

```
IF <condition expl 1>
    <statements>
[ELSEIF <condition expl 2>
    <statements>
[ELSEIF <condition expl 3>
    <statements>...]]
[ELSE
    <statements>]
ENDIF
```

<condition expl> A logical expression that determines if the set of statements between IF and the next ELSE, ELSEIF, IF, or ENDIF command execute. If the condition is true, the statements execute. If the condition is false, control passes to the next ELSE, ELSEIF, or ENDIF.

<statements> One or more program lines consisting of any combination of commands, functions, and preprocessor directives.

ELSEIF <condition expl> <statements> Specifies that when the previous IF or ELSEIF condition is false, control passes to this ELSEIF <condition expl>. As with IF, if the condition is true, only the set of statements between this ELSEIF and the next ELSEIF, ELSE, or ENDIF execute. If the condition is false, control passes to the next ELSEIF, ELSE, or ENDIF.

You can enter this option as either ELSEIF or ELSE IF. The ellipsis (...) in the syntax statement indicates that you can have multiple ELSEIF statements.

ELSE <statements> Specifies statements to execute if all previous conditions are false.

ENDIF A required command that marks the end of the IF structure.

Description

Use IF...ELSEIF...ENDIF to evaluate one or more conditions and execute only the set of statements following the first condition that evaluates to true. For the first true condition, dBASE executes the statements between that program line and the next ELSEIF, ELSE, or ENDIF, then skips everything else in the IF structure and executes the program line following ENDIF. If no condition is true and an associated ELSE command

exists, dBASE executes the set of statements after ELSE and then executes the program line following ENDIF.

Use IF...ENDIF to test one condition and IF...ELSEIF...ENDIF to test two or more conditions. If you have more than three conditions to test, consider using DO CASE instead of IF. Compare the example in this section with the example for DO CASE.

You can nest IF statements to test multiple conditions; however, the ELSEIF option is an efficient alternative. When you use ELSEIF, you don't need to keep track of which ELSE applies to which IF, nor do you have to put in an ending ENDIF.

You can put many commands in each set of commands. If the number of commands in a set makes the code hard to read, consider putting them in a procedure and calling the procedure from the IF statement instead.

Example

The following example of nested IF constructs determines the magnitude of a previously declared memory variable and displays an appropriate message. Contrast this code segment with the simpler ELSEIF construct of the second section:

```
nM_value=523
IF nM_value > 1000
    ? "Value is over 1000."
ELSE
    IF nM_value > 100
        ? "Value is over 100."
    ELSE
        IF nM_value > 10
            ? "Value is over 10."
        ELSE
            IF nM_value > 1
                ? "Value is over 1."
            ELSE
                ? "Value is 1 or less."
            ENDIF
        ENDIF
    ENDIF
ENDIF
```

The following example of a nested IF construct uses ELSEIF to determine the magnitude of a previously declared memory variable and displays an appropriate message:

```
IF nM_value > 1000
    ? "Value is over 1000."
ELSEIF nM_value > 100
    ? "Value is over 100."
ELSEIF nM_value > 10
    ? "Value is over 10."
ELSEIF nM_value > 1
    ? "Value is over 1."
ELSE
    ? "Value is 1 or less."
ENDIF
```

Portability

ELSEIF isn't supported in dBASE IV or dBASE III PLUS.

See Also

DO CASE, DO...UNTIL, DO WHILE, FOR...NEXT, IIF(), SCAN

IIF()**Programs**

Returns one of two values depending on the result of a specified logical expression.

Syntax

IIF(<expL>, <exp1>, <exp2>)

<expL> The logical expression to evaluate to determine whether to return <exp1> or <exp2>.

<exp 1> The character, date, logical, numeric, or float expression to return if <expL> evaluates to .T.

<exp 2> The character, date, logical, numeric, or float expression to return if <expL> evaluates to .F. The data type of <exp 2> doesn't have to be the same as that of <exp 1>.

Description

IIF() stands for "immediate IF" and is a shortcut to the IF...ELSE...ENDIF programming construct. Use IIF() where expressions are allowed but programming constructs aren't, such as in reports and labels, and to restrict a value to one of two possibilities.

Example

The following example uses IIF() to pass either of two results back to the variable or command using IIF():

```
x = IIF(SUBSTR(LDRIVER(),6,2)="US", "AMERICAN", "FRENCH")
SET DATE TO &x
? DATE()
```

The next example uses IIF() to index a table such that a selected state appears at the top of the ordered database rather than in its usual character or numeric value order.

```
USE Clients EXCLUSIVE
INDEX ON IIF(State_Prov="CA","A","Z") + STATE_PROV TAG TopState
LIST FIELDS Company, Contact, State_Prov OFF NEXT 30
CLOSE ALL
```

See Also

IF

Creates a dBASE table from data stored in files with different formats.

Syntax

```
IMPORT FROM <filename> | ?  
[[TYPE] WB1 | WK1]  
[HEADING]
```

<filename> | ? The name of the file you want to import. `IMPORT ?` displays a dialog box, from which you can select a file to import. If you specify a file without including its path, *Visual* dBASE looks for the file in the current directory, then in the path you specify with `SET PATH`.

[TYPE] WB1 | WK1 Specifies the format of data you are importing. The `TYPE` keyword is included for readability only; it has no effect on the operation of the command. The following table provides a description of the different file formats that are supported:

Type	Description
WB1	A Quattro Pro for Windows spreadsheet file. Spreadsheet rows form table records; spreadsheet columns form table fields. If you don't specify an extension, <i>Visual</i> dBASE assumes .WB1.
WK1	A Lotus 1-2-3 spreadsheet. Spreadsheet rows form table records; spreadsheet columns form table fields. If you don't specify an extension, <i>Visual</i> dBASE assumes .WK1.

[HEADING] Specifies using the label headings of spreadsheet columns as the new field names of the table created by `IMPORT`. Labels . Labels used as field names are truncated at 10 characters. Spaces are converted to the underscore character and other characters are mapped to valid dBASE field name characters.

Description

Use the `IMPORT` command to import data from files in non-dBASE formats. You need to specify the extension of a file, if it is something other than the default extension.

`IMPORT` creates the table in which data is imported in the same drive and directory as the original file and opens the table in the current work area. To display information about the imported file, use `DISPLAY STATUS` and `DISPLAY STRUCTURE`.

Example

The following example uses `IMPORT FROM` to create a new .DBF table from a Quattro Pro .WB1 file:

```
SET DBTYPE TO DBASE  
IMPORT FROM C:\QPW\SAMPLES\LOANPMT.WB1 TYPE WB1  
CD C:\QPW\SAMPLES  
USE LOANPMT  
BROWSE  
CLOSE ALL
```

See Also

APPEND, DISPLAY STATUS, DISPLAY STRUCTURE, USE

INDEX**Table organization**

Creates an index for the current table. For Paradox and SQL tables, you can define index tags, but you do not specify the name of an index file.

Syntax

```
INDEX ON <key exp>
[TO <.ndx filename> | ? | <.ndx filename skeleton>]
[UNIQUE]
```

or

```
INDEX ON <key exp> | <field list>
TAG <tag name>
    [OF <.mdx filename> | ? | <.mdx filename skeleton>]
    [FOR <condition>]
[DESCENDING]
[UNIQUE]
```

You can use a special form of this command, INDEX ON <field list> PRIMARY, to create a primary index on a Paradox table.

<key exp> | <field list> For dBASE tables, <key exp> specifies the name of a character, numeric, float, or date field, it can be a dBASE expression of up to 220 characters that includes operators or functions that manipulate field values, or a combination of field names and expressions. The maximum length of the key, the result of the evaluated index <key exp>, is 100 characters.

When <key exp> is based on more than one field, all elements of the expression must evaluate to the character type. You can join multiple fields or expressions using string concatenation operators (+ or -).

For Paradox and SQL tables, indexes can't include expressions: however, you can create indexes based on one or more fields. In that case, you specify the index key as a <field list>, separating the name of each field with a comma.

TAG <tag name> Specifies the name of the index tag added to an .MDX multiple index file. If you do not specify an .MDX file, index tags are added to the production .MDX file.

OF <.mdx filename 2> | ? | <.mdx filename skeleton> Specifies the .MDX multiple index file that dBASE adds new index tags to. OF ? and OF <filename skeleton> display a dialog box, in which you can select an existing .MDX file. If you specify a file that doesn't exist, Visual dBASE creates it and adds the index tag name. By default, Visual dBASE assigns an .MDX extension and saves the file in the current directory.

TO <.ndx filename 1> | ? | <.ndx filename skeleton> Specifies the name of an .NDX index file. By default, Visual dBASE assigns an .NDX extension to <filename 1> and saves the file in the

current directory. The ? and *<.ndx filename skeleton>* options display a dialog box, in which you specify the name of the target file and the directory to save it in.

FOR <condition> Restricts the records *Visual* dBASE includes in the index to those meeting the specified *<condition>*. Without the FOR *<condition>* or UNIQUE options, all records of the table are included in the index.

DESCENDING Creates the index in descending order (Z to A, 9 to 1, later dates to earlier dates). Without DESCENDING, INDEX creates an index in ascending order.

UNIQUE Prevents multiple records with the same *<key expC>* value from being included in the index; *Visual* dBASE includes in the index only the first record with that value. Without the UNIQUE or FOR *<condition>* options, all records of the table are included in the index. For SQL tables, specifies creating a unique index which prevents entry of duplicate index keys in a table.

Description

Use INDEX to organize data for rapid retrieval and ordered display. INDEX doesn't actually change the order of the records in a table but rather creates an index file in which records are arranged in numeric, float, character, or date order based on the value of a key expression. Like the index of a book, with ordered entries and corresponding page numbers, an index file contains ordered key expressions with corresponding record numbers. When the table is used with a master index, the contents of the table appear in the order specified by the index.

At the end of an indexing operation, the new index file is the master index, and the record pointer is positioned at the first record of the indexed table. *Visual* dBASE closes all other indexes except those whose tag names are in the production .MDX file, if one exists, with the same name as the table.

In an index, records are usually arranged in ascending order, with lowest key values at the beginning of the index. Using the DOS Code Page 437 (U.S.) character set, character keys are ordered in ASCII order (from A to Z and then from a to z); numeric and float keys are ordered from lowest to highest numbers; and date keys are ordered from earliest to latest date. Include the UPPER() function in the key expression to convert all lowercase letters to uppercase and achieve alphabetical order for character-type fields.

Note Most non-U.S. character sets provide a different sort order for characters than the DOS Code Page 437 character set.

You can reverse the order of an index, arranging records in descending order, by including the DESCENDING keyword. (You can use DESCENDING only when building .MDX tags.)

If a function is used in a key expression, keep in mind that the index is ordered according to the function output. Thus, when you use FIND or SEEK, or otherwise access the key value of a record, include the entire key expression. For example, INDEX ON SOUNDEX(Name) TO Names creates an index ordered by the values SOUNDEX() returns. When attempting to find data by the key value, you must include the entire key expression, such as SEEK SOUNDEX("Jones") rather than SEEK "Jones". Don't use functions such as CHR(), LTRIM(), RTRIM(), TRIM(), or IIF() that vary the field length in the key expression.

In some tables, multiple records may share the same *<key value>*. Use the UNIQUE option to include only the first such record in the index. INDEX with the UNIQUE option has the same effect as indexing a table with SET UNIQUE ON.

FOR *<condition>* limits the records whose key expression values dBASE includes in the index to those meeting the specified condition. For example, if you use INDEX ON Lastname + Firstname TO Salaried FOR Salary > 24000, *Visual* dBASE includes only records of employees with salaries higher than \$24,000 in the index. The FOR condition can't include calculated fields.

Once a table has been indexed, use LOOKUP(), SEEK, SEEK(), and FIND to retrieve data. The structure of an index file allows these commands to quickly locate values of the key expression.

When you execute APPEND, BROWSE, CHANGE, EDIT, INSERT, PACK, REPLACE, @...GET, or UPDATE, *Visual* dBASE automatically updates all open index files. Index files closed when changes are made in a table can be opened and then updated using REINDEX.

Multiple index files simplify updating indexes, since *Visual* dBASE updates all indexes with tag names listed in .MDX files specified with USE...ORDER or SET ORDER. *Visual* dBASE automatically opens a production .MDX file, if one exists, when you use the associated table.

INDEX...TAG creates an index and adds the tag name to a multiple index file. If you don't include OF *<filename>*, INDEX...TAG adds the tag name to the production .MDX file. *Visual* dBASE creates the production .MDX or the specified file if it doesn't already exist.

INDEX...TAG closes all open indexes except those with tag names in the production .MDX file, or within the same .MDX file you specify. If indexes with tag names listed in the specified .MDX file are not open, *Visual* dBASE opens them.

INDEX is similar to SORT, another command that allows ordering of a table. Unlike INDEX, though, SORT physically rearranges the table records, a time-consuming process for large files. To maintain the sorted order, either new records must be placed in their proper position using INSERT, or the entire table must be resorted. Also, SORT doesn't support SEEK, SEEK(), or FIND, making the process of locating data in a sorted table slower.

Example

The following example uses INDEX to create index tags for the current table:

```
USE Clients EXCLUSIVE
INDEX ON Company TAG Company
* creates an index by Company
BROWSE TITLE "Indexed by Company"
INDEX ON State_Prov+City ;
    TAG StateCity DESCENDING
* Combine State and City index
BROWSE TITLE "Indexed by State & City Descending"
INDEX ON City;
    TAG CA ;
```

```
INKEY( )
```

```
FOR State_Prov = "NY" && only cities in New York State  
BROWSE TITLE "Indexed by City, NY only"
```

Portability

The file-name skeleton option is not available in dBASE IV. TAG, FOR and .MDX options not available in dBASE III PLUS.

See Also

FIND, KEY(), LOOKUP(), ORDER(), REINDEX, SEEK, SEEK(), SET INDEX, SET ORDER, SET UNIQUE, SORT, TAG(), USE

INKEY()

Keyboard and mouse events

Returns the decimal value associated with the first key or key combination held in the keyboard typeahead buffer and removes the keystroke from the buffer. Can also be used to wait for a keystroke and return its value.

Syntax

```
INKEY([<seconds expN>] [, <mouse expC>])
```

<seconds expN> The number of seconds INKEY() waits for a keystroke. If <expN> is zero, INKEY() waits indefinitely for a keystroke.

<mouse expC> Determines whether INKEY() returns a value when you click the mouse. If <expC> is M or m, INKEY() returns -100. If <expC> is not M or m, INKEY() ignores a mouse click and waits for a keystroke.

Description

The keyboard typeahead buffer stores keystrokes the user enters while dBASE is busy processing other data. Use INKEY() to identify and delete a keystroke held in the keyboard typeahead buffer. If the typeahead buffer is empty, INKEY() returns the value of zero.

For example, if you press *C* and then *Alt+P*, dBASE stores the values 67 and -420 in the typeahead buffer. INKEY() returns 67, and a second INKEY() returns -420. See Appendix D for a complete list of the values returned by INKEY(). See Appendix E for a list of decimal values corresponding to keys.

To determine a value in the buffer in a position other than the first position, or to determine a buffer value without removing it from the buffer, use NEXTKEY().

Example

The following example continuously executes a loop that shows the value of INKEY() and the character typed. The loop ends when the Escape key (ASCII 27) is pressed:

```
CLEAR  
SET ESCAPE OFF  
* ESCAPE ON will interrupt the program and the  
* Escape key will not be trapped by INKEY()
```



```

? "Press Esc to continue"
k=0
DO WHILE k <> 27
    k=INKEY()
    ? k
    IF k>0
        ?? CHR(k)  && show the key and the ascii char
    ENDIF
ENDDO
SET ESCAPE ON

```

The following example displays a message and waits up to 10 seconds or until any key or mouse button is clicked:

```

? "This message will display for 10 seconds max"
Pause=inkey(10,"m")
IF Pause=0
    ? "You waited the 10 seconds"
ENDIF

```

I

Portability

The *<seconds expN>* option is not supported in dBASE III PLUS. The *<mouse expC>* option is not supported in dBASE IV or dBASE III PLUS. For some keys, *Visual* dBASE returns values different from those in earlier versions of dBASE. See Appendix D for a complete list of returned values.

See Also

CLEAR TYPEAHEAD, KEYBOARD, LASTKEY(), NEXTKEY(), ON KEY, READKEY(), SET TYPEAHEAD

INPUT

Input/Output

Accepts a user-entered expression and stores it to a memory variable. INPUT can accept character, numeric, float, date, or logical data. This command is supported primarily for compatibility with dBASE IV. In *Visual* dBASE, use DEFINE with the Text and EntryField classes for displaying and accepting information on a form.

For complete syntax information on INPUT, see online Help. For more information about working with *Visual* dBASE forms, see the Forms chapters in the *User's Guide*.

Syntax

INPUT [*<prompt expC>*] TO *<memvar>*

<prompt expC> A character expression that prompts the user for input.

TO <memvar> Assigns the user input to the memory variable you specify for *<memvar>*. If *<memvar>* doesn't exist, dBASE creates it. If *<memvar>* does exist, INPUT overwrites it.

Description

Use INPUT primarily in program files to prompt the user for these types of keyboard input:

- Non-character data
- Character data, entered in quotes or square brackets
- Complex expressions

The user terminates data entry by pressing Enter. If the user presses Enter without typing any characters, dBASE continues to prompt for input until the user enters something.

If SET ESCAPE is ON, pressing Esc in response to an INPUT terminates a program.

The type of expression the user inputs determines the type of memory variable INPUT creates:

Expression	Rules for entering	<memvar> variable type
Integer	Enter directly	Numeric
Character	Delimit with ' ' or " " or []	Character
Date	Delimit with { }	Date
Logical	.T. or .F.	Logical

The SET DECIMALS setting affects how numeric keyboard entries with decimals are displayed.

To ensure that the user enters the correct type of data, use <prompt expC> to indicate what the user should enter. For example, <prompt expC> could be "Enter today's date surrounded by braces ({ })". You could then use TYPE() to test the type of data the user enters.

Other commands that let programs read data from the keyboard are ACCEPT and WAIT.

Example

The following examples demonstrate the use of INPUT to enter different types of variables:

```
CLEAR
amount=0
INPUT"How much? " TO Amount
* Amount will be overwritten
? Amount,type("Amount")
INPUT"Enter a date (e.g. {01/01/01}) " TO Enterdate
? Enterdate, type("Enterdate")
INPUT 'Enter your name (e.g. "John") ' TO Name
? Name, type("Name")
```

Enterdate and Name are not guaranteed to contain a date and a name. They could contain .T., "abc", or 5.7, and be a logical, string, or numeric type.

See Also

ACCEPT, CTOD(), DEFINE SET DECIMALS, SET ESCAPE, TYPE(), WAIT

INSERT

Fields and records

Adds a new record to the current table at the current record location.

Syntax

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

BLANK Inserts a blank record, but doesn't display a window to update field values in the new record.

BEFORE When no master index is in use, inserts a new record before the current record rather than after.

NOWAIT Invokes the form to edit a new record but does not move focus there. If used in a program, execution continues to statement following the INSERT NOWAIT command.

Description

Use INSERT to add a single record within an existing table. If dBASE tables are linked with the SET RELATION command, the CONSTRAIN and INTEGRITY options control operations that add new records to child and parent tables. For more information, see the SET RELATION command.

When adding a record to an indexed table, INSERT adds the new record to the end of the table (with a record number one greater than the previous highest record number) and also correctly inserts the record in all open indexes. Index files closed at the time a new record is inserted in a table may be updated using REINDEX.

If you INSERT a record at the end of a file, INSERT (but not INSERT BLANK) functions like APPEND, continuing to add new records until you exit.

If SET CARRY is OFF, the default setting, newly inserted records are blank before editing. If SET CARRY is ON, each new record is filled with the contents of the preceding record.

INSERT BLANK adds a record to the current table and moves the record pointer to the new record, just as INSERT does, but doesn't display a window to update values in the blank record.

In a table without a master index, issuing the INSERT command without the BEFORE option adds a blank record immediately following the current record, also updating any open indexes. In a table with a master index, INSERT adds the record to the end of the table and updates all open indexes when you save the new record.

Example

The following example uses INSERT BLANK to insert a blank record after the current record:

```

SET TALK OFF
USE Clients EXCLUSIVE
SKIP 3
? RECNO(), Company      && Note recno() and company
SKIP-1                  && Moves record pointer back
INSERT BLANK
SKIP                    && Advance record pointer 1
? RECNO(), Company      && Former record is now 1;
                        record number higher

CLOSE ALL

```

The next example uses INSERT BEFORE to insert a blank record before the current record and present an entry window for user input.

```

USE Clients EXCLUSIVE
SKIP 4
? RECNO(), Company
INSERT BEFORE           && Edit window presented
? RECNO(), Company      && Newly added record in old;
                        record position

SKIP                    && Advance record pointer 1
? RECNO(), Company      && Old record

CLEAR
CLOSE ALL

```

See Also

APPEND, INSERT AUTOMEM, REINDEX, REPLACE, SET CARRY, SET FIELDS, SET RELATION, STORE, USE

INSERT AUTOMEM

Fields and records

Adds a new record to the current table using values stored in automem variables.

Syntax

```

INSERT AUTOMEM
[BEFORE]

```

BEFORE Inserts the new record before the current record.

Description

INSERT AUTOMEM adds a new record to a table using the values stored in automem variables. If no master index exists, INSERT AUTOMEM without the BEFORE option adds the record immediately after the current record. If a master index exists, INSERT AUTOMEM adds the record to the end of the table and updates all open indexes.

A full discussion of the use of automem variables for adding data to a table is included under APPEND AUTOMEM.

The only difference between INSERT AUTOMEM and APPEND AUTOMEM is that, in an unindexed table, INSERT AUTOMEM adds the record immediately after (or before) the current record, while APPEND AUTOMEM adds the record to the end of the table. INSERT AUTOMEM assigns the new record a record number one greater than (or less than) the current record and also changes all record numbers beyond the new record; APPEND AUTOMEM assigns the new record the last record number in the table.

When a master index is open, APPEND AUTOMEM and INSERT AUTOMEM work identically. Both assign the new record the last record number in the table and update all open indexes.

INSERT AUTOMEM provides advantages over INSERT BLANK since INSERT AUTOMEM updates a table only once, while INSERT BLANK updates a table first when it adds a blank record, and again when you use REPLACE to update the blank values.

Example

The following example seeks a specified record, uses STORE AUTOMEM to hold field names and values for the current record, checks to determine the presence of a Collect table and uses INSERT AUTOMEM to transfer the selected record's data to the Collect table:

```
CLEAR
SET TALK OFF
USE Clients ORDER Client_ID
Lookup = "A3367"
SEEK Lookup
IF FOUND()
    STORE AUTOMEM
ELSE
    RETURN
ENDIF
IF .NOT. FILE("Collect.DBF")
    COPY STRUCTURE TO Collect
ENDIF
USE Collect EXCLUSIVE
INSERT AUTOMEM
BROWSE
CLOSE ALL
SET TALK ON
RETURN
```

Portability

Not supported in dBASE IV or dBASE III PLUS.

See Also

APPEND AUTOMEM, INSERT

Opens the Inspector, a window that lists object properties and lets you change their settings.

Syntax

INSPECT(<*object reference*>)

<*object reference*> A reference to the object that you want to inspect. dBASE generates a variable containing an object reference when you create an object with the NEW operator or the DEFINE command. This variable has the same name you gave to the object.

Description

Use INSPECT() to examine and change object properties directly. For example, during program development you can use INSPECT() to evaluate objects and experiment with different property settings.

The Inspector is modeless, and doesn't affect program execution.

Note You can access the Inspector from the Form Designer by right-clicking the form or one of its objects and selecting Inspector from the SpeedMenu.

You can get help on any property in the Inspector by selecting the property and pressing **F1**.

Example

The following example uses INSPECT() to open a dialog box that displays the current properties of spinbox Spin1 after changing the spinbox value:

```
PUBLIC F1
USE COUNTRY
SET PROCEDURE TO PROGRAM(1) ADDITIVE
DEFINE FORM F1 FROM 2,2 TO 15,40;
    PROPERTY Text "SpinBoxDemo"
DEFINE SPINBOX Spin1 OF F1;
    PROPERTY;
        Datalink "Country->GNP",;
        Top 2,;
        Left 4,;
        Height 2,;
        Step 1000
DEFINE PUSHBUTTON Proceed OF F1;
    PROPERTY;
        OnClick Check,;
        Top 5,;
        Left 6,;
        TEXT "Proceed"
OPEN FORM F1

PROCEDURE Check
    ? INSPECT(F1.Spin1)
```

CLOSE FORMS F1
RETURN

Portability

Not supported in dBASE IV or dBASE III PLUS.

See Also

DISPLAY MEMORY, DISPLAY STATUS

INT()

Numeric data

Returns the integer portion of a specified number.

Syntax

INT(<expN>)

<expN> A numeric or float number whose integer value to determine and return.

I

Description

Use INT() to remove the decimal digits of a numeric or float number and retain only the integer portion, the whole number. INT() returns an integer whose data type is numeric.

If you pass a number with decimal places to a function or command that uses an integer as an argument, such as _pageno or SLEEP, dBASE automatically truncates that number, in which case you don't need to use INT().

The following table compares INT(), FLOOR(), CEILING(), and ROUND(). (In these examples, the value of the second ROUND() argument is 0.)

<expN>	INT()	FLOOR()	CEILING()	ROUND()
2.56	2	2.00	3.00	2.60
-2.56	-2	-3.00	-2.00	-2.60
2.54	2	2.00	3.00	2.50
-2.54	-2	-3.00	-2.00	-2.50

Example

The following example uses INT() to display the contents of a field:

```
SET TALK OFF
CLEAR
STORE 0 TO rate, min_time, total
STORE "U" TO up_down
STORE "$ " TO set_cur
@ 4, 8 SAY "What currency do you want to use " + ;
    "($, DM, FR, YEN)" GET set_cur FUNCTION "@"
@ 5,32 SAY "What is the billing rate?" ;
    GET rate PICTURE "999.99"
@ 6,19 SAY "What is the billing time (in minutes)?";
```

ISALPHA()

```
GET min_time PICTURE "9999"
@ 7,25 SAY "Do you want to round Up or Down?" ;
GET up_down PICTURE "!" VALID up_down $ "UD"
@ 8,12 SAY "What is the total billable time (in minutes)?";
GET total PICTURE "9999"
READ

time = INT(total/60) + (MOD(total,60)/60)
bill = IIF(up_down = "U", CEILING(total/min_time), FLOOR(total/min_time))
@ 10,12 SAY "The total billable time is " + LTRIM(STR(time,7,2)) + " hours"
dec = SET("DECIMALS")
SET DECIMALS TO 2
cur = SET("CURRENCY")
SET CURRENCY TO TRIM(set_cur)
IF set_cur = "YEN"
    SET CURRENCY RIGHT
ENDIF
IF set_cur = "DM" .OR. set_cur = "FR"
    sep = SET("SEPARATOR")
    point = SET("POINT")
    SET SEPARATOR TO "."
    SET POINT TO ","
ENDIF
@ 11,12 SAY "For a total of "
@ 11,27 SAY bill * rate PICTURE "$999,999.99"
SET DECIMALS TO dec
SET CURRENCY TO cur
IF set_cur = "YEN"
    SET CURRENCY LEFT
ENDIF
IF set_cur = "DM" .OR. set_cur = "FR"
    SET SEPARATOR TO sep
    SET POINT TO point
ENDIF
```

See Also

ABS(), CEILING(), FLOOR(), ROUND()

ISALPHA()

String data

Returns .T. if the first character of a string is alphabetic.

Syntax

ISALPHA(<expC> | <memo field>)

<expC> | <memo field> The character string or memo field to test.

Description

ISALPHA() tests the first character of a character expression or memo field and returns .T. if it's an alphabetic character. ISALPHA() returns .F. if the character isn't alphabetic or if the character expression or memo field is empty.

The current language driver defines the character values that are lowercase and uppercase alphabetic. In a U.S. language driver, a lowercase alphabetic character is from a to z, and an uppercase alphabetic character is from A to Z. See Appendix C in the *Programmer's Guide* for more information about language drivers.

Example

The following example uses ISALPHA() to determine if the first character of each string is an alphabetic character:

```
? ISALPHA("Visual dBase")           && Returns .T.
? ISALPHA(" Visual dBase")          && Returns .F.
? ISALPHA("")                       && Returns .F.
? ISALPHA('2548 Vestal Parkway')    && Returns .F.
```

The next example uses ISALPHA() to determine whether a character location specified by SUBSTR() is alphabetic. With the variable Is_Alpha initialized to .F., the Street procedure steps through the contents of Address field until it encounters an alphabetic character and returns the string that follows:

```
CLEAR
SET TALK OFF
USE Clients
DO WHILE .NOT. EOF()
    ? Street(Address)           && just the street name
    SKIP
ENDDO

FUNCTION Street
Parameter Full_Addr
Full = TRIM(Full_Addr)
Len = LEN(Full)
Start = 0
IF Len > 0                     && Check if longer than 0
    Is_Alpha = .F.
    Char_Pos = 1
    DO WHILE .NOT. Is_Alpha    && Check until alpha
        Is_Alpha = ISALPHA(SUBSTR(Full,Char_Pos,1))
        Start = Start + 1
        Char_Pos = Char_Pos + 1
    ENDDO
    Street = SUBSTR(Full,Start,Len-Start)
ELSE
    Street = ""
ENDIF
RETURN Street                 && Return the street name only
```

Portability

The <memo field> argument isn't supported in dBASE IV or dBASE III PLUS.

See Also

CHARSET(), ISLOWER(), ISUPPER(), LDRIVER(), LOWER(), UPPER()

ISBLANK()

Determines if a specified field or expression is blank.

Syntax

ISBLANK(<exp>)

<exp> An expression of any data type.

Description

ISBLANK() returns .T. if a specified expression is blank, .F. if it contains data. A field is blank if it has never contained a value or if you used the BLANK command on it.

ISBLANK() returns a different result from EMPTY() when used on numeric fields;

ISBLANK() differentiates between zero and blank values, while EMPTY() does not.

ISBLANK() is especially useful when performing functions such as averaging, since it ensures that blank values are not included in the calculation. If you don't need to differentiate between 0 or blank values in numeric fields, you can use either ISBLANK() or EMPTY().

Example

The following interactive commands from the Command window demonstrate the functionality of ISBLANK():

```

Empty = SPACE(20)
? ISBLANK(Empty)           && Returns .T.
Empty = " "
? ISBLANK(Empty)           && Returns .T.
mDate = { / / }           && or {}
? ISBLANK(mDate)           && Returns .T.
USE Clients
APPEND BLANK               && adds new blank record
? ISBLANK(StartBal)        && Returns .T.
REPLACE StartBal WITH 0
? ISBLANK(StartBal)        && Returns .F.
BLANK FIELDS StartBal
? ISBLANK(StartBal)        && Returns .T.
? ISBLANK(Notes)           && Returns .T. for memo
REPLACE Notes WITH "Something for the memo field"
? ISBLANK(Notes)           && Returns .F. for memo field

```

Use ISBLANK() to exclude blank records from calculations.

```
CALCULATE AVG(StartBal) FOR .NOT. ISBLANK(StartBal)
```

The following example uses ISBLANK() to select only those records with non-blank values in the StartBal field of the Clients table to create a report:

```

SET SAFETY OFF
SET TALK OFF
USE Clients EXCLUSIVE
INDEX ON Company TAG Company
BLANK FIELDS StartBal FOR StartBal = 0

```

```

? CENTER("Start Balance Report")
?
SCAN
  IF .NOT. ISBLANK(StartBal)
    ? COMPANY + "Balance Due " + ;
      TRANSFORM(StartBal, "@$999,999,999.99")
  ENDIF
ENDSCAN
RETURN

```

See EMPTY() for additional examples of ISBLANK().

Portability

Not supported in dBASE III PLUS.

See Also

APPEND, BLANK, EMPTY(), SPACE(), TYPE()

I

ISCOLOR()

Colors and fonts

Returns .T. if system monitor is color, .F. if it is monochrome. ISCOLOR() is supported primarily for compatibility with dBASE IV. *Visual* dBASE determines the color palette to use from the Color settings specified in the Windows Control Panel.

For more information about ISCOLOR(), see online Help. For more information about Windows colors, see your Windows documentation.

ISLOWER()

String data

Returns .T. if the first character of a string is alphabetic and lowercase.

Syntax

ISLOWER(<expC> | <memo field>)

<expC> | <memo field> The character string or memo field to test.

Description

ISLOWER() tests the first character of a character expression or memo field and returns .T. if it's a lowercase alphabetic character. ISLOWER() returns .F. if the character isn't lowercase or if the character expression or memo field is empty.

The current language driver defines the character values that are lowercase and uppercase alphabetic. In a U.S. language driver, a lowercase alphabetic character is from a to z, and an uppercase alphabetic character is from A to Z. See Appendix C in the *Programmer's Guide* for more information about language drivers.

ISLOWER()

Example

The following example uses ISLOWER() to determine if the first character of each string is a lowercase letter:

```
? ISLOWER("Visual dBase")           && Returns .F.
? ISLOWER(" Visual dBase")          && Returns .F.
? ISLOWER("")                        && Returns .F.
? ISLOWER('2548 Vestal Parkway')    && Returns .F.
? ISLOWER("software craftsmanship") && Returns .T.
```

Portability

The *<memo field>* argument isn't supported in dBASE IV or dBASE III PLUS.

See Also

CHARSET(), ISALPHA(), ISUPPER(), LDRIVER(), LOWER(), UPPER()

ISMOUSE()

Keyboard and mouse events

Returns a logical true (.T.) when a mouse driver is currently installed on your system, or false (.F.) when no mouse driver is present.

Syntax

ISMOUSE()

Description

Use ISMOUSE() to determine if a mouse driver is installed on the current system. For example, you may want to use ISMOUSE() to control whether a message says "Click OK to continue" or "Choose OK to continue."

Example

The following example uses ISMOUSE() to determine if a mouse is enabled and returns the appropriate message to the Command window results pane:

```
IF ISMOUSE()
  ? "The mouse is enabled" AT 5
ELSE
  ? "The mouse is disabled" AT 5
ENDIF
```

Portability

Not supported in dBASE III PLUS.

See Also

INKEY(), MDOWN(), MCOL(), MROW(), ON MOUSE

ISTABLE()

Table basics

Tests for the existence of a table in a specified database and returns .T. if the table exists or .F. if it doesn't.

Syntax

ISTABLE(<table name>)

<table name> The name of the table to search for. You must also provide the full path if the table is not in the current directory or a directory specified with SET PATH. You can also specify a path relative to the current directory.

You can also check for the existence of a table in a database (defined using the BDEBDE Configuration Utility) by specifying the database as a prefix (enclosed in colons) to the name of the table, that is, *:database name:table name*. If the database is not already open, Visual dBASE displays a dialog box in which you specify the parameters, such as a login name and password, necessary to establish a connection to that database.

I

Description

Use ISTABLE() to confirm the existence of a table of the type specified by SET DBTYPE. If the table is not in the current default directory, you must include the directory path with the name of the table. You can also specify a database if you want to search for a table that is not in the current database.

Example

The following example uses ISTABLE() to check for the existence of a specified Paradox table on the SAMPLES directory, and if present, opens the table in a Browse:

```
CLOSE ALL
CLEAR
IF ISTABLE("C:\VISUALDB\SAMPLES\Customer.DB")
    USE Customer
    BROWSE
ELSE
    ? "No such table exists"
ENDIF
RETURN
```

See Also

DIR, DISPLAY FILES, FILE(), GETFILE(), PUTFILE(), SET DEFAULT, SET DATABASE, SET DBTYPE, SET DIRECTORY, SET PATH

ISUPPER()

Returns .T. if the first character of a string is alphabetic and uppercase.

Syntax

ISUPPER(<expC> | <memo field>)

<expC> | <memo field> The character string or memo field to test.

Description

ISUPPER() tests the first character of a character expression or memo field and returns .T. if it's an uppercase alphabetic character. ISUPPER() returns .F. if the character isn't uppercase or if the character expression or memo field is empty.

The current language driver defines the character values that are lowercase and uppercase alphabetic. In a U.S. language driver, a lowercase alphabetic character is from a to z, and an uppercase alphabetic character is from A to Z. See Appendix C in the *Programmer's Guide* for more information about language drivers.

Example

The following example uses ISUPPER() to determine if the first character of each string is an uppercase letter:

```
? ISUPPER("Visual dBase")           && Returns .T.
? ISUPPER(" Visual dBase")          && Returns .F.
? ISUPPER("")                       && Returns .F.
? ISUPPER('2548 Vestal Parkway')    && Returns .F.
? ISUPPER("Software Craftmanship")  && Returns .T.
```

Portability

The <memo field> argument isn't supported in dBASE IV or dBASE III PLUS.

See Also

CHARSET(), ISALPHA(), ISLOWER(), LDRIVER(), LOWER(), UPPER()

ITOH()

Returns the hexadecimal equivalent of a specified decimal number as a character string.

Syntax

ITOH(<expN 1>[, <expN 2>])

<expN 1> The decimal number whose hexadecimal equivalent to return.

<expN 2> The number of characters to include in the returned hexadecimal character string. If <expN 2> is greater than the number of characters returned, ITOH() pads the returned string with leading 0's to make it <expN 2> characters long. For example, ITOH(21) returns the string "15", while ITOH(21,4) returns "0015".

If expN2 is smaller than the length of the returned string, it is ignored.

Description

Use ITOH() to convert a decimal number to a character string representing its hexadecimal equivalent. ITOH() is the inverse function of HTOI(), which accepts hexadecimal numbers in the form of character expressions and returns a decimal equivalent. Use both functions with Windows Application Programming Interface (API) calls that require hexadecimal values.

Example

```
? ITOH(13824,4)  && Returns 3600
```

Portability

Not supported in dBASE III PLUS or dBASE IV.

See Also

HTOI()

J

JOIN

Table organization

Combines records of the current table with records of a specified table to create a new table.

Syntax

```
JOIN WITH <alias> TO <filename> | ? | <filename skeleton>
[[TYPE] PARADOX | DBASE]
FOR <condition>
[FIELDS <field list>]
```

<alias> The alias table with which to combine the current table's records to create <filename>.

TO <filename> | ? | <filename skeleton> Creates the table file <filename>. By default, *Visual* dBASE assigns a .DBF extension to <filename> and saves the file in the current directory. The ? and <filename skeleton> options display a dialog box, in which you specify the name of the target file and the directory to save it in.

You can also create a table in a database (defined using the BDE Configuration Utility) by specifying the database as a prefix (enclosed in colons) to the name of the table, that is, *:database name:table name*. If the database is not already open, *Visual* dBASE displays a dialog box, in which you specify the parameters, such as a login name and password, necessary to establish a connection to that database.

[[TYPE] PARADOX | DBASE Specifies the type of table you want to create. The TYPE keyword is included for readability only; it has no effect on the operation of the command.

Specifying DBASE creates a dBASE table (the default). If you don't include an extension for <filename>, *Visual* dBASE assigns a .DBF extension.

Specifying PARADOX creates a Paradox table with a .DB extension.

FOR <condition> Restricts JOIN to records in the current table that meet the specified <condition>.

FIELDS <field list> Limits the fields of the new table to those included in <field list>. Enter field names from the current table directly and field names of the alias table in the form *alias->field*. Without FIELDS, the new table contains all fields of both tables.

Description

Use JOIN to create new tables joining the records of two existing tables. The fields list can specify any type of field from either table, except for binary, memo, and OLE fields.

You can use the SET FIELDS command before JOIN, rather than specifying a fields list. In that case, only the fields listed in SET FIELDS are included in the new table. If you do not specify a fields list, field assignments are first made from the current table and then from the second table. If both tables contain a field with the same name, only the field from the first table is added to the new table; the field in the second table is ignored.

The FOR condition typically compares the values of fields in records of one table with the value of fields in records of another table. In the simplest case, the equality operator (=) determines whether the values of one table's fields are identical to the other table's fields (regardless of whether the field names are the same). Use care when formulating the FOR condition to ensure that only the records you want are included in the new table, as the JOIN command can potentially create a very large table (equal to the number of records in the current table multiplied by the number of records in the alias table).

Because of the number of records compared during the JOIN operation, JOIN is also potentially a very time-consuming command. To provide temporary combinations of data from more than one table, without taking the time required by JOIN, use the SET RELATION command.

Example

The following example uses JOIN to create a new table with specified fields drawn from both the original tables based on a stated relation between the two tables:

```
CLOSE DATABASE
USE Company IN SELECT() && Work area 1
USE Contact IN SELECT() && Work area 2
SELECT Contact
JOIN WITH Company ;
    FOR Contact->CompCode = Company->CompCode ;
    FIELDS Contact->CompCode, Company->Company, Contact->Contact;
    TO CompCntc
```

The CompCntc table has been created with a structure of the fields in the FIELDS clause. There is now a single table with fields Compcode, Company, and Contact. The FOR clause was necessary to ensure that each company was linked to the contacts at that company.


```
SELECT 3
USE CompCntc
LIST OFF
```

See Also

SELECT, SET RELATION, USE

KEY()

Table organization

Returns the key expression used to create the specified index.

Syntax

KEY([<*mdx filename*>] <*index position expN*> [, <*alias*>])

<*mdx filename*> Specifies a multiple index file that contains the index tag you want to check.

<*index position expN*> Selects an index file or tag by the position of an index tag in an .MDX file or the position of an index file in the list of open indexes for the current or a specified work area.

<*alias*> A work area number (1 through 225), letter (A through J), or alias name. The work area letter or alias name must be enclosed in quotes.

K

Description

The KEY() function returns the key expression that was used to create a specified index. If you have several open .NDX indexes or several indexes with tag names listed in multiple index files, you can use KEY() to determine the key expression of each index.

If no index is open in the current or specified work area, or if <*index position expN*> doesn't evaluate to the position of any index in the index list, KEY() returns an empty string ("").

Example

The following example uses KEY() to retrieve the indexes key statement.

```
USE Company EXCLUSIVE
INDEX ON CompCode TAG CompCode
INDEX ON Zip_Postal+Company TAG ZipCompany

TagCompCode=TAGNO("CompCode")
* Get the Tag number of CompCode
TagZipCompany=TAGNO("ZipCompany")
* Get the Tag number of ZipCompany

? TagCompCode, TAG(TagCompCode), KEY(TagCompCode)
? TagZipCompany, TAG(TagZipCompany), ;
  KEY(TagZipCompany)
```

Portability

Not supported in dBASE III PLUS.

See Also

INDEX, NDX(), ORDER(), SET INDEX, SET ORDER, TAG(), TAGCOUNT(), TAGNO(), USE

KEYBOARD

Keyboard and mouse events

Puts the value of <expC> into the typeahead buffer. This command is supported primarily for compatibility with dBASE IV. In *Visual dBASE*, forms do not use the typeahead buffer.

For complete syntax information on KEYBOARD, see online Help. For information about working with forms, see the Forms chapters in the *User's Guide*.

KEYMATCH()

Table organization

Indicates if a specified expression is found in an index.

Syntax

KEYMATCH (<exp> [,<index position expN> |
[<.mdx filename expC>,<tag expN>]
[,<alias>]])

<exp list> Specifies the expression of any data type that you want to look for. For Paradox and SQL tables, you can specify one or more values (separated by commas) that match single or composite index key fields.

<index position expN> Specifies an .NDX file by the position of the index in the list of open indexes for the current or a specified table.

<.mdx filename expC> Specifies a multiple index file that contains the index tag you want to check.

<tag expN> Specifies an index tag by the position of an index tag in an .MDX file for the current or a specified table.

<alias> Specifies the work area where the specified .NDX or .MDX is open. You can specify a work area number (1 through 225), letter (A through J), or alias name. The work area letter or alias name must be enclosed in quotes.

Description

The KEYMATCH() function determines if specified key expressions are found in a particular index. KEYMATCH() returns .T. or .F. to indicate whether the specified expression was found. SET EXACT controls whether exact matches of character string data is required.

A primary use of the KEYMATCH() function is to check for duplicate values during an APPEND operation.

KEYMATCH() looks only in the specified index file or tag. It ignores the settings for SET DELETED, SET FILTER, and SET KEY, ensuring the integrity of data in a table even when you work with a subset of the table records.

If you specify only an expression (<exp>) whose value you want to match, KEYMATCH() searches the current master index for an index key with the same value. If a matching index key is found, KEYMATCH() returns .T.

Example

The following example uses KEYMATCH() to determine if a passed value is found in the index:

```
SET EXACT OFF && KEYMATCH is affected by SET EXACT
USE Company EXCLUSIVE
INDEX ON Company TAG CompanyCa FOR State_Prov = "CA"
INDEX ON City TAG City OF Location
? KEYMATCH("Cons",TAGNO("CompanyCa"))
? KEYMATCH("Compton","Location",TAGNO("City"))
```

Both answers return .T. from the sample table.

Portability

Not supported in dBASE III PLUS.

L

See Also

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

LABEL FORM

Input/Output

Generates and displays or prints a label report, using the label format stored in a specified label file and information derived from records in the current table.

Syntax

```
LABEL FORM <filename 1> | ? | <filename skeleton 1>
[<scope>] [FOR <condition 1>] [WHILE <condition 2>]
[SAMPLE]
[TO FILE <filename 2> | ? | <filename skeleton 2>] | [TO PRINTER]
```

<filename 1> | ? | <filename skeleton> The file to get label formats from. The ? and <filename skeleton> options display a dialog box from which you can select a file. If you specify a file without including its path, dBASE looks for the file in the current directory, then in the path you specify with SET PATH. If you specify a file without including its extension, dBASE looks for .RPL, .LBG, or .LBL, in that order.

<scope> The number of records in the current table from which to derive labels. RECORD <n> identifies a single record by its record number. NEXT <n> identifies n

records, beginning with the current record. ALL specifies all records. REST specifies all records from the current record to the end of the file.

FOR <condition 1>

WHILE <condition 2> Determines which records are affected by LABEL FORM. FOR restricts LABEL FORM to records that meet <condition 1>. WHILE starts processing with the current record and continues with each subsequent record as long as <condition 2> is true.

SAMPLE Displays or prints a label containing asterisks for text and then prompts you for more samples.

TO FILE <filename 2> | ? | <filename skeleton 2> Directs output to the text file <filename>. By default, dBASE assigns a .TXT extension to <filename> and saves the file in the current directory. The ? and <filename skeleton> options display a dialog box in which you specify the name of the target file and the directory to save it in.

TO PRINTER Directs output to the printer.

Description

Use LABEL FORM to print or display labels in a format that you've defined in the Report Designer using CREATE LABEL or MODIFY LABEL. For information about using the Report Designer, see the Crystal Reports documentation. If you don't specify a <scope>, WHILE <condition 1>, or FOR <condition 2> option, LABEL FORM prints the label specifications for each record in record number or index order.

When printing or displaying a label report that includes groups of data or group subtotals, either the current table must be in sorted order or its master index must be in use. The sorted file or index must be arranged according to the value of the field on which the data is grouped.

LABEL FORM without the TO FILE or TO PRINTER options displays the labels in the results pane of the Command window or current user-defined window.

Example

This example opens the Company database and then generates labels using the Complbl1 label form:

```
CLOSE DATABASE
USE Company
LABEL FORM Complbl1 TO PRINT
* This label format produces one across labels,
* 6 lines per label, with Company, Street,
* City, State and Zip code:
* General Consolidated
* 35 Libra Plaza
* Nashua NH 09242
*
*
* Consolidated Brands, Inc.
* 3 Independence Parkway
* Rivendell CA 93456
```

Portability

The *<filename skeleton>* option is not supported in dBASE IV or dBASE III PLUS.

See Also

CREATE LABEL

LASTKEY()

Keyboard and mouse events

Returns the value of the key or key combination that was pressed to terminate execution of a full-screen command. This command is supported primarily for compatibility with dBASE IV. In *Visual* dBASE, use properties such as OnClick to initiate actions based on how a user exits a form.

For complete syntax information on LASTKEY(), see online Help. For information about working with forms, see the Forms chapters in the *User's Guide*.

LDRIVER()

Environment

Returns the name of the language driver the current table or a specified table is using. If no table is open and you issue LDRIVER() without an argument, it returns the global language driver in use.

L

Syntax

LDRIVER([*<alias>*])

<alias> A work area number (1 through 225), letter (A through J), or alias name. The work area letter or alias name must be enclosed in quotes.

Description

Use LDRIVER() to learn which language driver the current table or a specified table is using. If you don't pass LDRIVER() an argument, it returns the name of the language driver of the current table or, if no tables are open, the global language driver in use. LDRIVER() also returns information on Paradox and SQL databases.

The language driver associated with a table depends on the DOS code page or the BDE language driver setting that was in effect when the table was created. With *Visual* dBASE, you can choose the language driver that applies to your dBASE data in the [CommandSettings] section in the DBASEWIN.INI file. For example, you can load a German language driver to work with a table created while that driver was active.

Example

This example shows the LDRIVER() function and a sample response:

```
? LDRIVER()  && DB437US0
```

This example first closes all tables and obtains the global language driver. Then it opens a table and checks whether the table was created with the global language driver. If not, a warning is displayed:

LEFT()

```
CLOSE ALL
SET LDCHECK OFF
* this program replaces the LDCHECK alert message
GlobalDriver=LDRIVER()
USE Customer
TableLangDriver=LDRIVER()
SET EXACT ON
IF GlobalDriver<>TableLangDriver
  ? "Warning: this table was created"+ "with a different language driver"
  ? "Global Language Driver: "+GlobalDriver
  ? DBF()+" Language Driver: "+TableLangDriver
WAIT
ENDIF
SET EXACT OFF
```

Portability

Not supported in dBASE IV or dBASE III PLUS.

See Also

ANSI(), CHARSET(), OEM(), SET LDCHECK

LEFT()

String data

Returns a specified number of characters from the beginning of a character string or memo field.

Syntax

LEFT(<expC> | <memo field>, <length expN>)

<expC> | <memo field> The string or memo field to extract characters from.

<length expN> The number of characters to extract from the beginning of the string or memo field.

Description

Starting with the first character of a character expression or a memo field, LEFT() returns a specified number, <length expN>, of characters. LEFT() returns a maximum of 32766 characters, the maximum length of a string.

If <length expN> is greater than the number of characters in the specified string or memo field, LEFT() returns the string as it is, without adding space characters to achieve the specified length. You can use LEN() to determine the actual length of the returned string.

If <length expN> is less than or equal to zero, LEFT() returns an empty string. If <length expN> is greater than or equal to zero, LEFT(<expC>, <length expN>) achieves the same results as SUBSTR(<expC>, 1, <length expN>).

When LEFT() returns characters from a memo field, it counts two characters for each carriage-return and linefeed combination (CR/LF).

Example

The following example uses LEFT() to return a portion of a text string, starting from the left end of the string:

```
? LEFT("dBASE",1)           && Returns "d"
? LEFT("dBASE",3)           && Returns "dBA"
? LEFT("dBASE",9)           && Returns "dBASE"
? LEFT("dBASE",0)           && Returns ""
```

The next example uses ISALPHA() and SUBSTR() to determine the character position of the first alpha character in a string (Address). The derived variable (To) is then used as the length parameter of LEFT() to display only the street number portion of the Address field:

```
CLOSE DATABASES
SET TALK OFF
CLEAR
USE Clients
DO WHILE .NOT. EOF()
? Street_No(Address)           && Returns just
                               && the street number

SKIP
ENDDO
CLOSE DATABASES

FUNCTION Street_No
Parameter Full_Addr
Full = TRIM(Full_Addr)
Len = LEN(full)
To = 0
IF Len > 0                     && Check if longer
    Is_Alpha = .F.             && than 0
    Char_Pos = 1
    DO WHILE .NOT. Is_Alpha    && Check til alpha
        Is_Alpha = ISALPHA(SUBSTR(full,char_pos,1))
        IF .NOT. Is_Alpha      && Add to variable
            To = To + 1        && to if ISALPHA()
        ENDIF                 && returns .F.
        Char_Pos = Char_Pos + 1
    ENDDO
    Street = LEFT(Full,To-1)
ELSE
    Street = ""
ENDIF
RETURN Street                 && Return the
                               && street number
```

L

Portability

The *<memo field>* argument isn't supported in dBASE III PLUS. Both dBASE III PLUS and dBASE IV limit the return value of LEFT() to 254 characters.

See Also

AT(), LEN(), RIGHT(), SUBSTR()

Returns the number of characters in a specified character string or memo field.

Syntax

LEN(<expC> | <memo field>)

<expC> | <memo field> The character string or memo field whose length to find.

Description

LEN() returns the number of characters (the *length*) of a character string or memo field. The length of an empty character string or empty memo field is zero. LEN() counts an embedded null character, CHR(0), as one character. When LEN() calculates the length of a memo field, it counts two characters for each carriage-return and linefeed combination (CF/LF).

To find the length of numeric or float data, use LENNUM().

To find the number of lines in a memo field, use MEMMLINES(). To find the length of a particular line of a memo field, use LEN() with MLINE(). For example, LEN(MLINE(Descrip,3)) returns the length of the third line in the memo field Descrip.

Example

The following examples use LEN() to determine the length of text strings:

```
? LEN("Aloha!")           && Returns 6
? LEN("")                  && Returns 0
? LEN("Hello" + " There")  && Returns 11
```

The next example uses LEN() to determine the number of characters in the Notes memo field. If Notes has no contents, LEN() returns 0:

```
USE Clients
SCAN
  ? IIF(LEN(Notes)>0,Notes,"Record ";
  + LTRIM(STR(RECNO()))+" has no Notes")
ENDSCAN
CLOSE DATABASES
```

Portability

The <memo field> argument isn't supported in dBASE III PLUS, and dBASE III PLUS and dBASE IV don't count null characters.

See Also

LENNUM(), MEMMLINES(), MLINE(), TRIM()

LENNUM()

Numeric data

Returns the display length of a specified number, including leading spaces.

Syntax

LENNUM(<expN>)

<expN> The numeric or float number whose display length to return.

Description

Use LENNUM() before formatting a display involving numeric values of varying lengths.

If you pass LENNUM() the name of a numeric field, it returns the length of the field.

If a number has eight or fewer whole-number digits and no decimal point, it is by default a numeric-type number; the default display length for numeric-type numbers is 10. For example, LENNUM(123) returns 10.

Example

The following example uses LENNUM() to determine the character length of a number:

```
CLEAR
dec = SET("DECIMALS")
FOR x = 3 TO 9 STEP 1
    SET DECIMALS TO x
    num_val = 3.2 * 1.53
    ? "The results of 3.2 * 1.53 - " + STR(num_val,7 + x,x) + " - is " + ;
      LTRIM(STR(LENNUM(num_val),2,0)) + " characters in length"
    ? " when DECIMALS is SET TO " + LTRIM(STR(x,1,0))
    ?
NEXT
SET DECIMALS TO dec
```

L

Portability

Not supported in dBASE III PLUS or dBASE IV. In dBASE III PLUS and dBASE IV.

See Also

LEN(), SET DECIMALS, STR()

LIKE()

String data

Returns .T. if a specified string matches a specified skeleton string.

Syntax

LIKE(<skeleton expC>, <expC> | <memo field>)

<skeleton expC> A string containing a combination of characters and wildcards. The wildcards are ? and *.

<expC> | <memo field> The string or memo field to compare to the skeleton string.

Description

Use LIKE() to compare one string to another. The <skeleton expC> argument contains wildcard characters and represents a pattern; the <expC> or <memo field> argument is compared to this pattern. LIKE() returns .T. if <expC> or <memo field> evaluates to a string that matches <skeleton expC>. To compare the phonetic similarity between two strings rather than the character-by-character similarity, use DIFFERENCE().

Use the wildcard characters ? and * to form the pattern for <skeleton expC>. An asterisk (*) stands for any number of characters, including zero characters. The question mark (?) stands for any single character. Both wildcards can appear anywhere and more than once in <skeleton string>. Wildcard characters in <skeleton expC> can stand for uppercase or lowercase letters.

If * or ? appears in <expC> or <memo field>, they are interpreted as literal, not wildcard, characters, as shown in the following example.

```
LIKE("a*d", "abcd") && returns .T.
LIKE("a*d", "aBCd") && returns .T.
LIKE("abcd", "a*d") && returns .F.
```

LIKE() is case-sensitive. Use UPPER() or LOWER() for case-insensitive comparisons with LIKE(). This is shown in the following example.

```
LIKE("*xyz", "uvwxyz") && returns .T.
LIKE("*xyz", "UVWXYZ") && returns .F.
LIKE(LOWER("*xyz"), LOWER("UVWXYZ")) && returns .T.
```

LIKE() returns .T. if both arguments are empty strings. LIKE() returns .F. if one argument is empty and the other isn't.

LIKE() isn't affected by SET EXACT or by the current language driver.

Example

The following example uses LIKE() to determine whether two text strings are similar:

```
? LIKE("abc", "abc")           && Returns .T.
? LIKE("abc", "Abc")           && Returns .F.
? LIKE("a?c", "abc")           && Returns .T.
? LIKE("a*c", "abc")           && Returns .T.
? LIKE("a*", "abc")             && Returns .T.
? LIKE("*bc", "abc")           && Returns .T.
? LIKE("?abc", "abc")          && Returns .F.
```

The next example uses LIKE() to list only those records that contain "COMPUTER" in the company field:

```
USE Clients
LIST FIELDS Company, Contact ;
  FOR LIKE("*COMPUTER", UPPER(Company))
CLOSE DATABASES
```

Portability

Not supported in dBASE III PLUS. The *<memo field>* argument isn't supported in dBASE IV.

See Also

AT(), DIFFERENCE(), LDRIVER(), LOWER(), RAT(), SET EXACT, SUBSTR(), UPPER()

LINENO()

Error handling and debugging

Returns the number of the current program line in the current program, procedure, or user-defined function (UDF).

Syntax

LINENO()

Description

Use LINENO() to track program flow. Use it in conjunction with PROGRAM() to learn when a program executes a given line of code. You can also use LINENO() with ON ERROR to find out which line produces an error.

LINENO() is meaningful only when issued from within a program, procedure, or UDF. When issued in the Command window, LINENO() returns 0.

LINENO() always returns the actual program line number; the number doesn't reflect the order in which the line executes within the program.

Example

See ON ERROR for an example of using LINENO().

Portability

Not supported in dBASE III PLUS.

See Also

ERROR(), MESSAGE(), PROGRAM(), RESUME, SUSPEND

L

LIST

Syntax

```
LIST
[<scope>]
[FOR <condition 1>]
[WHILE <condition 2>]
[[FIELDS] <exp list>]
[OFF]
[TO FILE <filename> | ? | <filename skeleton>] | [TO PRINTER]

LIST COVERAGE
[<.COV filename> | ? | <filename skeleton>]
[ALL]
[SUMMARY]
[TO FILE <filename> | ? | <filename skeleton>] | [TO PRINTER]

LIST FILES
[[LIKE] <filename 1> | <filename skeleton 1>]
[ON <drive>]
[TO FILE <filename 2> | ? | <filename skeleton 2>] | [TO PRINTER]

LIST MEMORY
[TO FILE <filename> | ? | <filename skeleton>] | [TO PRINTER]

LIST STATUS
[TO FILE <filename> | ? | <filename skeleton>] | [TO PRINTER]

LIST STRUCTURE
[IN <alias>]
[TO FILE <filename> | ? | <filename skeleton>] | [TO PRINTER]
```

Table organization

Error Handling and
Debugging

Disk and file management

Environment

Environment

Table basics

Description

All LIST commands have equivalent DISPLAY commands that output the same information; both LIST and DISPLAY output to the results pane of the Command window. The only difference is that LIST commands list continuously, halting after the last window of information, while DISPLAY commands start with and pause at the first window of information. See DISPLAY for a description of how to navigate through DISPLAY and LIST output in the results pane. See the complementary DISPLAY commands for descriptions of the information each DISPLAY and LIST command produces.

If the information output to the results pane is more than the *Visual* dBASE buffer can contain, you might not be able to scroll back up to information you've scrolled down through. Use the TO FILE or TO PRINTER options to save all information to a file or as printer output.

Example

The following example uses LIST to show selected data from the Clients table:

```
USE Clients EXCLUSIVE
LIST ALL
* list all fields in all records

GO TOP      && begin at the first record
LIST NEXT 10 FIELDS Company, Contact OFF
* list only Company and Contact fields
* in the next 10 records
* OFF turns off the record number

INDEX ON COMPANY TAG COMPANY
* create an index by Company
SEEK "C" && find the first Company beginning with "C"
LIST COMPANY WHILE COMPANY="C" OFF
* show Company names so long as the Company begins
* with the letter "C"
```

See Also

DISPLAY, DISPLAY COVERAGE, DISPLAY FILES, DISPLAY MEMORY, DISPLAY STATUS, DISPLAY STRUCTURE

LISTCOUNT()

Objects

L

Returns the number of prompts in a list box.

Syntax

LISTCOUNT(<form reference>.<list box reference>)

<form reference>.<list box reference> *<form reference>* is an object reference variable pointing to the form in which the list box is placed. *<list box reference>* is an object reference variable pointing to the list box you evaluate.

You can create *<form reference>* and *<list box reference>* with the DEFINE command:

```
* Create an object reference variable, MyForm.
DEFINE FORM MyForm
* Create an object reference variable, xChoose.
DEFINE LISTBOX xChoose OF MyForm
```

You can also create *<form reference>* and *<list box reference>* with the NEW operator:

```
MyForm = NEW FORM()
xChoose = NEW LISTBOX("MyForm")
```

Description

Use LISTCOUNT() when you can't anticipate the number of prompts a list box may have at run time. For example, when you specify "FILE *.*" for the DataSource property, the number of prompts varies when files are added or deleted from the default directory.

LISTCOUNT()

You can use LISTCOUNT() to control loops that evaluate user choices in a multiple-choice list box. For example, you can see which prompts were chosen by evaluating each prompt with the LISTSELECTED() function in a DO...WHILE loop.

You make a list box multiple-choice by setting the Multiple property to true (.T.).

Example

The following example defines a form that contains a listbox that displays names from the ANIMALS.DBF table. Property Multiple .T. provides that the user can select more than one listbox prompt. The OnRightMouseDown property calls procedure Checked, which uses LISTCOUNT() and LISTSELECTED() to send the selected prompts to the Command window results pane with each OnRightMouseDown:

```
LOCAL f
f = NEW GFORM()
f.Open()

CLASS GFORM OF FORM
    this.Left =          58.60
    this.Height =        10.12
    this.Width =          41.00
    this.Text = "Animals of the World"
    this.OnRightMouseDown = CHECKED
    this.HelpId = ""
    this.HelpFile = ""
    this.Top =           9.35

    DEFINE LISTBOX LB1 OF THIS;
        PROPERTY;
            Left          9.00;;
            Height        4.00;;
            ColorNormal "N/W*";
            Width         20.00;;
            DataSource "FIELD ANIMALS->NAME";
            ColorHighLight "W+/B";
            Multiple .T.;;
            Top           3.00

ENDCLASS

PROCEDURE Checked
FOR i=1 TO LISTCOUNT(Form.LB1)
    ? LISTSELECTED(Form.LB1,i)
NEXT i
RETURN
```

Portability

Not supported in dBASE IV or dBASE III PLUS.

See Also

DO...WHILE, FOR...NEXT, LISTSELECTED(), Multiple

LISTSELECTED()

Objects

Returns a list box prompt.

Syntax

LISTSELECTED(<form reference>.<list box reference> [, expN])

<form reference>.<list box reference> <form reference> is an object reference variable pointing to the form in which the list box is placed. <list box reference> is an object reference variable pointing to the list box you evaluate.

You can create <form reference> and <list box reference> with the DEFINE command:

```
* Create an object reference variable, MyForm.
DEFINE FORM MyForm
* Create an object reference variable, xChoose.
DEFINE LISTBOX xChoose OF MyForm
```

You can also create <form reference> and <list box reference> with the NEW operator:

```
MyForm = NEW FORM()
xChoose = NEW LISTBOX("MyForm")
```

<expN> The number of the prompt to evaluate. If you include <expN>, LISTSELECTED() returns the prompt of the specified item only if it is selected. If you omit <expN>, LISTSELECTED() returns the currently-chosen prompt in a single-choice list box or the most recently chosen prompt in a multiple-choice list box.

L

Description

Use LISTSELECTED() to evaluate which list box prompt or prompts are selected.

Use LISTSELECTED() in conjunction with LISTCOUNT() to evaluate user choices in a multiple-choice list box. For example, you can determine which prompts were chosen by evaluating each prompt with the LISTSELECTED() function in a DO...WHILE loop. (If the list box has an uncertain number of prompts, you use LISTCOUNT() to determine the number of times to execute the loop.)

You make a list box multiple-choice by setting the Multiple property to true (.T.).

Example

The following example defines a form that contains a listbox that displays names from the ANIMALS.DBF table. Property Multiple .T. provides that the user can select more than one listbox prompt. The OnClose procedure Checked uses LISTCOUNT() and LISTSELECTED() to send the selected record prompts to the Command window results pane with OnRightMouseDown:

```
LOCAL f
f = NEW GFORM()
f.Open()

CLASS GFORM OF FORM
    this.Left = 58.60
    this.Height = 10.12
    this.Width = 41.00
```

LKSYS()

```
this.Text = "Animals of the World"
this.OnRightMouseDown = CHECKED
this.HelpId = ""
this.HelpFile = ""
this.Top = 9.35

DEFINE LISTBOX LB1 OF THIS;
PROPERTY;
Left 9.00;;
Height 4.00;;
ColorNormal "N/W*";;
Width 20.00;;
DataSource "FIELD ANIMALS->NAME";;
ColorHighLight "W+/B";;
Multiple .T.;;
Top 3.00

ENDCLASS

PROCEDURE Checked
FOR i=1 TO LISTCOUNT(Form.LB1)
? LISTSELECTED(Form.LB1,i)
NEXT i
RETURN
```

Portability
Not supported in dBASE IV or dBASE III PLUS.

See Also
DO...WHILE, FOR...NEXT, LISTCOUNT()

LKSYS()

Shared data

Returns information about a locked record or file.

Syntax
LKSYS(<expN>)

<expN> A number representing the information for LKSYS() to return:

<expN>	Returns
0	Time when lock was placed
1	Date when lock was placed
2	Login name of user who locked record or file
3	Time of last update or lock
4	Date of last update or lock
5	Login name of user who last updated or locked record or file

Description

LKSYS() returns multiuser information contained in a `_dbaselock` field of a table. For LKSYS() to return information, the current table must have a `_dbaselock` field. Use CONVERT to add a `_dbaselock` field to a table. If the current table doesn't contain a `_dbaselock` field, LKSYS() returns an empty string for any value of `<expN>`.

When a record is locked, either explicitly or automatically, the time, date, and login name of the user placing the lock are stored in the `_dbaselock` field of the record. When a file is locked, this same information is stored in the `_dbaselock` field of the first record of the table.

Passing 0, 1, or 2 as arguments to LKSYS() returns values only after an attempted file or record lock has failed. If a file or record lock on a converted table file fails, the information for LKSYS() arguments 0, 1, and 2 is written to a buffer from the table's `_dbaselock` field. If you then pass 0, 1, or 2 to LKSYS(), the information is read from the buffer. The buffer isn't overwritten until you attempt another lock that fails. Thus, 0, 1, and 2 always return the information that was current at the time of the last lock failure.

You can pass 3, 4, or 5 as arguments to LKSYS() whether or not the current record or file is currently locked. These arguments return information about the last successful record or file lock. When you pass any of these arguments to LKSYS(), it returns information directly from the `_dbaselock` field rather than from an internal buffer.

If you pass 2 or 5 to obtain a user login name, and the `_dbaselock` field is only 8 characters wide, LKSYS() returns an empty string. The first 8 characters of a `_dbaselock` field are the count, time, and date information of the last update or lock, so the field must be wider than 8 characters to fit part or all of the login user name. Set the width of the field with CONVERT.

Note LKSYS() doesn't return locking information about SQL databases or Paradox tables.

Example

The following example uses RLOCK() to lock a record of the Company table. If not successful, LKSYS() returns time, date and user id information on the network user who has locked the record. If the lock is successful the user will branch to a data entry procedure:

```
CLEAR
USE Company SHARED
GoTo 2
mRetry="Y"
DO WHILE UPPER(mRetry)="Y"
  IF .NOT. RLOCK()
    LckTime = LKSYS(0)
    LckDate = LKSYS(1)
    LckUser = LKSYS(2)
    ? "Lock Info: ",LckTime, LckDate, LckUser
    ?
    Wait "Try again? (Y/N)" to mRetry
  ELSE
    ? "Lock successful - proceed with data entry"
    mRetry=""
  ENDIF
```

L

```
ENDDO  
?  
WAIT
```

Portability

Not supported in dBASE III PLUS.

See Also

CHANGE(), CONVERT, FLOCK(), RLOCK(), SET LOCK, UNLOCK

LOAD DLL

Windows programming

Initiates a DLL file.

Syntax

LOAD DLL [*<path>*] *<DLL filename>*

[*<path>*] *<DLL name>* The name of the DLL file. *<path>* is the directory path to the DLL file in which the external function is stored.

Description

Use LOAD DLL to make the resources of a DLL file available to your application.

You can also use LOAD DLL to check for the existence of a DLL file. For example, you can use the ON ERROR command to execute an error trapping routine each time the LOAD DLL command can't find a specified DLL file.

LOAD DLL does not use the dBASE path to find DLL files. Instead, it searches the following directories:

- 1 The current default directory
- 2 The Windows directory (the directory containing WIN.COM)
- 3 The Windows system directory (the directory containing GDI.EXE)
- 4 The directory containing DBASEWIN.EXE
- 5 The directories listed in the DOS environment variable PATH
- 6 The directories mapped in a network (if any)

A DLL file is a precompiled library of external routines written in non-dBASE languages such as C and Pascal. A DLL file can have any extension, although most have extensions of .DLL.

When you initialize a DLL file with LOAD DLL, dBASE can access its resources; however, it doesn't become resident in memory until your program or another Windows program uses its resources.

To access a DLL function, create a dBASE function prototype with EXTERN. Then, using the name you specified with EXTERN, call the routine as you would any dBASE function.

LOAD DLL also loads and registers VBX controls.

Example

The following example uses LOAD DLL to initialize an image resource from a .DLL file:

```
LOAD DLL MyPicts.DLL
DEFINE FORM Pics FROM 2,2 TO 20,40
DEFINE Image MyPict OF Pics;
  PROPERTY DataSource "Resource MyPicts.DLL 1001", Top 5, Left 5
OPEN FORM Pics
```

Portability

Not supported in dBASE IV or dBASE III PLUS.

See Also

EXTERN, RELEASE DLL

LOCAL

Memory variables

Declares memory variables that are visible only in the subroutine where they're declared.

Syntax

LOCAL <memvar list>

<memvar list> The list of memory variables to declare local.

Description

Use LOCAL to declare a list of memory variables available only to the subroutine in which the command is issued. Local variables differ from those declared PRIVATE in one way; private variables are available to lower-level subroutines, while local variables are not. Local variables are accessible *only* to the subroutine—the program, procedure, or user-defined function (UDF)—in which they are declared.

A variable that is local to a subroutine is in effect a different variable from one with the same name in a higher-level or unrelated routine. Once a local variable is assigned a value, DISPLAY MEMORY and LIST MEMORY indicate a variable of the same name in a higher-level subroutine as hidden.

To declare a variable LOCAL, do so before initializing it to a particular value. Declaring a variable LOCAL, however, doesn't create it. After declaring a variable LOCAL, you can create and initialize it to a value with STORE or =. You can't declare local arrays. Local variables are erased from memory when the subroutine that creates them finishes executing.

L

For more information, see PUBLIC for a table that compares the scope and availability of public, private, local, and static variables.

Example

The following example uses LOCAL to declare the variable Today as a memory variable available only in Procedure Results:

```
SET PROCEDURE TO PROGRAM(1) ADDITIVE
DEFINE FORM Tenure FROM 0,0 TO 15,40
DEFINE TEXT T1 OF Tenure AT 3,4;
    PROPERTY Text "Start Date? (Month/Day/Year)", Width 30
DEFINE Entryfield F1 OF Tenure AT 3,28;
    PROPERTY Value {}, Picture "99/99/99", Width 9
DEFINE EntryField F2 OF Tenure AT 5,28;
    PROPERTY OnGotFocus Results, Value " ", Width 9
DEFINE TEXT T2 OF Tenure AT 5,4;
    PROPERTY TEXT "Years with the company", Width 22
OPEN FORM Tenure

PROCEDURE Results
Local Today
Today = Date()
Form.F2.Value = (Today-Form.F1.Value)/365
RETURN
```

Portability

Not supported in dBASE IV or dBASE III PLUS.

See Also

CLEAR MEMORY, PRIVATE, PUBLIC, RELEASE, STATIC, STORE

LOCATE

Table organization

Searches a table for the first record that matches a specified condition.

Syntax

LOCATE

[<scope>]

[FOR <condition 1>]

[WHILE <condition 2>]

<scope> The number of records to locate. RECORD <n> identifies a single record by its record number. NEXT <n> identifies *n* records, beginning with the current record. ALL specifies all records. REST specifies all records from the current record to the end of the file.

FOR <condition 1>

WHILE <condition 2> Determines which records are affected by LOCATE. FOR restricts LOCATE to records that meet <condition 1>. WHILE starts processing with the current record and continues with each subsequent record as long as <condition 2> is true.

Description

LOCATE performs a sequential search of a table and tests each record for a match to the specified conditions. If a match is found (FOUND() returns .T.), the record pointer of the table is positioned at that record. Entering CONTINUE resumes a search, allowing additional records meeting the specified condition to be found.

If no records meet the specified conditions or the bottom of the table is reached, *Visual dBASE* returns the message **End of LOCATE scope** (if SET TALK is ON) and positions the record pointer at the end of file (EOF() returns .T., and FOUND() returns .F.). *Visual dBASE* also returns this message if entering CONTINUE fails to find a record meeting the condition specified in the previous LOCATE.

LOCATE ALL and LOCATE FOR <condition> begin a search at the beginning of the current table, regardless of the position of the record pointer. However, LOCATE WHILE <condition>, LOCATE NEXT <n>, and LOCATE REST begin the search with the current record, rather than with the first record.

LOCATE does not require an indexed table; however, if an index is in use, LOCATE follows its index order. LOCATE uses the rules established by SET EXACT to determine whether a record meets the specified conditions. If SET EXACT is OFF (the default setting), only the beginning characters of the string on the right side of an equals sign need to match the string entered on the left side for LOCATE to determine that a condition has been met. If SET EXACT is ON, the strings must be identical to meet the condition.

The search commands LOCATE, SEEK, and FIND are each designed for use under particular conditions. LOCATE is the most flexible, accepting expressions of any data type as input and searching any field of a table. For large tables, however, a sequential search using LOCATE might be slow.

Use FIND or SEEK for greater speed. Both conduct an indexed search, similar to looking up a topic in a book index and turning directly to the appropriate page, allowing information to be found almost immediately. Once you use the INDEX command to create an index for a table, FIND and SEEK use this index to quickly identify an appropriate record. SEEK offers greater flexibility than FIND by accepting dBASE expressions as well as character and numeric input in specifying values of the key expression.

Example

The following example uses LOCATE to find the first company in the file that is in Texas:

```
USE Clients EXCLUSIVE
LOCATE FOR State_Prov="TX"
* locate does not depend on the index
IF FOUND()
    DISPLAY FIELDS Company, City, State_Prov
ELSE
    ? "No companies in Texas"
ENDIF
```

L

See Also

CONTINUE, FIND, FOUND(), LOOKUP(), SEEK, SEEK(), SET EXACT

LOCK()

Shared data

Locks the current record or a specified list of records in the current table or a specified alias table, and returns .T. if successful.

Syntax

LOCK([<record list expC>] | [<bookmark list expC>][,<alias>])

<list expC> The list of record numbers to lock, separated by commas.

<bookmark list expC> The list of bookmarks (record indicators) returned by BOOKMARK() specifying a record in a non-dBASE table, such as a Paradox table, that doesn't have natural record-order record numbers. Separate bookmarks by commas.

<alias> A work area number (1 through 225), letter (A through J), or alias name. The work area letter or alias name must be enclosed in quotes. If you don't include <alias>, LOCK() acts on the current table.

You don't have to specify record numbers or bookmarks if you want to specify a value for <alias>. However, if you have specified record numbers or bookmarks, you must precede <alias> with a comma (,).

Description

LOCK() is equivalent to RLOCK(). For more information, see RLOCK().

Example

See RLOCK() which is identical to LOCK().

Portability

Not supported in dBASE III PLUS. The <bookmark list expC> option isn't supported in dBASE IV.

See Also

FLOCK(), RLOCK(), SET LOCK, SET RELATION, SET REPROCESS, UNLOCK

LOG()

Numeric data

Returns the logarithm to the base e (natural logarithm) of a specified number.

Syntax

LOG(<expN>)

<expN> A positive nonzero number that equals e raised to the log. If you specify 0 or a negative number for <expN>, dBASE returns an error.

Description

LOG() returns the natural logarithm of *<expN>*, expressed as a float number. The natural logarithm is the power (exponent) to which you raise the mathematical constant *e* to get *<expN>*. For example, LOG(5) returns 1.61 because $e^{1.61}=5$.

LOG() is the inverse of EXP(). For example, if LOG(Y)=X, then Y=EXP(X).

Use SET DECIMALS to set the number of decimal places LOG() displays.

Example

The following example uses LOG() to return the base value and uses EXP() to find the cube roots for a set of passed values:

```
CLEAR
SET DECIMALS TO 2
? "Values" AT 7, "Log" AT 20, "Exponential of cube" AT 29
? "root of Log values" AT 29
FOR Value = 45 TO 270 STEP 15
? Value AT 3, LOG(Value) AT 12, EXP(LOG(Value)/3) AT 28
NEXT
```

See Also

EXP(), LOG10(), SET DECIMALS

LOG10()

Numeric data

L

Returns the logarithm to the base 10 of a specified number.

Syntax

LOG10(*<expN>*)

<expN> A positive nonzero number which equals 10 raised to the log. If you specify 0 or a negative number for *<expN>*, dBASE returns an error.

Description

LOG10() returns the common logarithm of *<expN>*, expressed as a float number. The common logarithm is the power (exponent) to which you raise 10 to get *<expN>*. For example, LOG10(100) returns 2 because $10^2=100$.

Use SET DECIMALS to set the number of decimal places LOG10() displays.

Example

The following example uses LOG10() to return base 10 values:

```
SET DECIMAL TO 2
?
? "Values" AT 7, "Base 10" AT 19, "Exponential of base 10 values" AT 29
FOR value = 50 TO 200 STEP 10
@ ROW() + 1, 3 SAY value
@ ROW(),13 SAY LOG10(value)
```

LOGOUT

```
@ ROW(),23 SAY LOG10(EXP(value))  
NEXT
```

Portability

Not supported in dBASE III PLUS.

See Also

EXP(), LOG(), SET DECIMALS

LOGOUT

Security

LOGOUT logs out the current user and sets up a new log-in dialog.

Syntax

LOGOUT

Description

LOGOUT logs out the current user from the current session and sets up a new log-in dialog 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, a log-in dialog appears. The user can 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 tables, their associated files, and program files.

If PROTECT has not been used, and no DBSYSTEM.DB file exists, the LOGOUT command is ignored.

See also

PROTECT, QUIT

LOOKUP()

Table organization

Searches a field for a specified expression and, if the expression is found, returns the value of a field within the same record.

Syntax

LOOKUP(<return field 1>, <exp>, <lookup field 2>)

<return field 1> The field of the current or specified table whose value you want to return if a match is found.

<exp> The expression to look for in the <lookup field 2>. Specify an alias when referring to fields outside the current work area.

<lookup field 2> The field of the current or a specified table whose value must match **<exp>**. If **<return field 2>** is a memo field, **<exp>** must be a character expression.

Description

The LOOKUP() function searches for values of **<lookup field 2>** that match the specified expression **<exp>**. If it finds a match, LOOKUP() positions the record pointer to the first record where a match occurred and returns the value of **<return field 1>**.

If no match is found, LOOKUP() returns an empty string (""), 0, an empty date, or .F., depending on the data type of **<lookup field 1>**. Also, EOF() returns .T., FOUND() returns .F., and the record pointer is positioned at the end of the file.

LOOKUP() performs a sequential search, unless an index whose key matches **<return field 2>** is open as a master index. To minimize the time LOOKUP() takes to search a table, you can specify a master index.

When you use LOOKUP() after executing a SET RELATION command, the **<exp>** and **<lookup field 2>** can be in the parent table, and **<return field 1>** can be in the child table.

Example

The following example uses LOOKUP() to find the first company in Chicago:

```
USE Company
Answer=LOOKUP(Company,"Chicago",City)
* Answer now contains the name of a company in
* Chicago or else is empty
IF .NOT. EMPTY(Answer)
  ? "In Chicago :"+ Answer
ELSE
  ? "No one in Chicago"
ENDIF
```

L

The following example takes each record in the Company table and uses LOOKUP() to look up in the Contact table the name of a contact at that company:

```
CLOSE DATA
USE CONTACT
SELECT 2
USE Company EXCLUSIVE
SCAN
  ? Company,LOOKUP(Contact->Contact,;
                    Compcode,Contact->Compcode)
ENDSCAN
```

Portability

Not supported in dBASE III PLUS.

See Also

EOF(), FOUND(), LOCATE, SEEK, SEEK(), SET EXACT, SET INDEX, SET ORDER

LOWER()

Converts all uppercase characters in a string to lowercase and returns the resulting string.

Syntax

LOWER(<expC> | <memo field>)

<expC> | <memo field> The character string or memo field to convert to lowercase.

Description

LOWER() converts the uppercase alphabetic characters in a character expression or memo field to lowercase. LOWER() ignores digits and other characters. LOWER() returns a maximum of 32766 characters, the maximum length of a string.

The current language driver defines the character values that are lowercase and uppercase alphabetic. In a U.S. language driver, a lowercase alphabetic character is from a to z, and an uppercase alphabetic character is from A to Z. See Appendix C in the *Programmer's Guide* for more information about language drivers.

Example

The following example uses LOWER() to convert uppercase text to lowercase:

```
? LOWER("Technical")      && Returns "technical"
? LOWER("")               && Returns ""
? LOWER("12 APPLES")      && Returns "12 apples"
```

In a field containing names in the format FIRSTNAME (Space) LASTNAME all in capital letters, UPPER() and LOWER() can be used to change the name string to upper- and lowercase as follows:

```
USE Contact
REPLACE ALL Contact WITH UPPER(SUBSTR(Contact,1,1))+;
    LOWER(SUBSTR(Contact,2,AT(" ",Contact)-2))+ " " +;
    UPPER(SUBSTR(Contact,AT(" ",Contact)+1,1))+;
    LOWER(SUBSTR(Contact,AT(" ",Contact)+2,16))
```

While this example demonstrates the use of UPPER(), LOWER(), and SUBSTR() to manipulate text strings, the previous task could also be accomplished by the following command.

```
REPLACE ALL Contact with PROPER(Contact)
```

Portability

The <memo field> argument isn't supported in dBASE IV or dBASE III PLUS.

See Also

CHARSET(), ISALPHA(), ISLOWER(), ISUPPER(), LDRIVER(), PROPER(), UPPER()

LTRIM()

String data

Returns a string with no leading space characters.

Syntax

LTRIM(<expC> | <memo field>)

<expC> | <memo field> The string or memo field to remove the leading space characters from.

Description

LTRIM() returns a character expression or memo field with no leading space characters. LTRIM() returns a maximum of 32766 characters, the maximum length of a string.

Using LTRIM() with a memo field removes leading spaces only at the beginning of the first line of the field. To remove leading spaces from a particular line of a memo field, use LTRIM() with MLINE().

To remove *trailing* space characters from a string or memo field, use RTRIM() or TRIM().

Example

The following example uses LTRIM() to remove leading spaces from text in a character field:

```
? STR(11.95,8,2)           && Returns 11.95
? LTRIM(STR(11.95,8,2))    && Returns 11.95
```

Numeric data defaults to the right in a table while character data defaults left. Further, when numeric data is stored to a memory variable, the variable defaults to 10 spaces long. The following example uses LTRIM() and STR() to left-justify numeric data:

```
USE Clients
96                && Positions record pointer at record number 96
X=Startbal       && store contents of Startbal to X
? X              && Returns 56.36
Y=LTRIM(STR(X,13,2))
? Y              && Returns 56.36
```

Portability

The <memo field> argument isn't supported in dBASE IV or dBASE III PLUS, and dBASE IV and dBASE III PLUS limit the return value of LTRIM() to 254 characters.

See Also

LEFT(), MLINE(), RIGHT(), RTRIM(), STR(), SUBSTR(), TRIM()

L

LUPDATE()

Returns the date of the last change to the current or a specified table.

Syntax

LUPDATE([<alias>])

<alias> A work area number (1 through 225), letter (A through J), or alias name. The work area letter or alias name must be enclosed in quotes.

Description

LUPDATE() displays the last update date of the specified table. If no table is open, LUPDATE() returns a blank date. 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.

Example

The following example uses LUPDATE() to return the date that a specified table was last updated:

```
USE Company IN SELECT()
USE Contact IN SELECT()
? "The table - Company - was last updated on " + DTOC(LUPDATE("Company"))
? "The table - Contact - was last updated on " + DTOC(LUPDATE("Contact"))
CLOSE ALL
```

See Also

DTOC(), SET CENTURY, SET DATE

MAX()

Compares two expressions of the same data type and returns the greater value. If comparing two logical expressions, returns .T. if one or both expressions are true.

Syntax

MAX(<exp 1>, <exp 2>)

<exp 1> An expression (character, date, logical, float, or numeric) to compare to a second expression of the same data type. You can compare float to numeric type numbers.

<exp 2> An expression of the same data type as <exp 1> to compare to <exp 1>. You can compare float to numeric type numbers.

Description

Use MAX() to compare two numbers, two character strings, two dates, or two logical values to determine the greater of the two values compared. MAX() returns:

- The greater of two numbers
- The character string with the higher *collation value* of two character strings. Collation values are determined by the language driver in use. For more information on language drivers, see Appendix C in the *Programmer's Guide*.
- The later of two dates
- True if one or both of two logical expressions evaluate to true

If <exp 1> and <exp 2> are equal, MAX() returns their value. For example, if you issue STORE 100 TO mvar1 and STORE 50+50 TO mvar2, MAX(mvar1,mvar2) returns 100.

If you use MAX() with character strings, first issue SET EXACT ON to ensure accurate results. When comparing character strings, MAX() is case-sensitive.

Example

The following example uses MAX() to compare a YTD_Sales field value with a memory variable that holds a computed average and returns the greater value:

```
CLEAR
SET TALK OFF
USE Company
AVERAGE YTD_Sales FOR YTD_Sales<>0 TO Midpt
GO TOP
? CENTER("Subsidiaries Exceeding Corporate Average Sales")
?
? "Company", "Subsid Avg" AT 27, "YTD Sales" AT 42, "Excess" AT 56
? REPLICATE(" ",7) AT 0, REPLICATE(" ",10) AT 27, ;
  REPLICATE(" ",9) AT 42, REPLICATE(" ",6) AT 56
DO WHILE .NOT. EOF()
  IF MAX(YTD_Sales,MidPt)=Ytd_Sales
    ? Company, Midpt AT 25, YTD_Sales, YTD_Sales - Midpt
  ENDIF
  SKIP
ENDDO
RETURN
CLOSE ALL
```

M

Portability

In dBASE III PLUS, MAX() supports only numeric expressions. In dBASE IV, MAX() doesn't support logical expressions.

See Also

CALCULATE, IIF(), MIN(), LDRIVER(), SET EXACT, SET LDCHECK

MCOL()

Keyboard and mouse events

Returns the current column position of the mouse pointer. This command is supported primarily for compatibility with dBASE IV. In *Visual* dBASE, use properties such as OnLeftMouseDown to manage mouse actions in forms.

For complete syntax information on MCOL(), see online Help. For information about working with forms, see the Forms chapters in the *User's Guide*.

MD

Disk and file utilities

Creates a new DOS directory.

Syntax

MD <directory>

<directory> The directory you want to create.

Description

MD and MKDIR are equivalent commands. See the description of MKDIR for more information.

Example

See MKDIR for an example of MD.

Portability

Not supported in dBASE IV or dBASE III PLUS.

See Also

CD, MKDIR, SET DIRECTORY

MDOWN()

Keyboard and mouse events

Returns .T. if the mouse button is pressed, .F. if it is not. This command is used primarily in a non-event driven environment. In *Visual dBASE*, use properties such as OnLeftMouseDown to manage mouse actions in forms.

For complete syntax information on MDOWN(), see online Help. For information about working with forms, see the Forms chapters in the *User's Guide*.

MDX()

Table organization

Returns the name of an .MDX file open in the current or a specified work area.

Syntax

MDX(<index tag expN>[, <alias>])

<index tag expN> The position of an index tag in the open multiple index file whose name you want to return.

<alias> A work area number (1 through 225), letter (A through J), or alias name. The work area letter or alias name must be enclosed in quotes.

Description

MDX() returns the name of the .MDX file open in the current or specified work area that contains a specified index tag. The index tag position corresponds to the .MDX file position in the index file list specified by SET INDEX TO, SET ORDER TO, or USE commands. If SET FULLPATH is ON, MDX() also returns the drive and directory location of the .MDX file in addition to its name.

If you do not specify an index order number, MDX() returns the name of the .MDX file that contains the master index tag. If you do not specify an index order number and the master index is an .NDX file, MDX() returns an empty string (""). MDX() also returns an empty string if there is no .MDX file open or no index tag in the specified position.

Example

The following example uses MDX() to determine the .MDX files that are open for the current table:

```
USE Company EXCLUSIVE
? 1,MDX(1)
? 2,MDX(2)
? " Only one MDX is open"
INDEX ON City      TAG City      OF Location
? 1,MDX(1)
? 2,MDX(2)
? "Now two MDXs are open"
```

M

Portability

Not supported in dBASE III PLUS.

See Also

FOR(), INDEX, NDX(), SET FULLPATH, SET INDEX, SET ORDER, TAG(), TAGCOUNT(), TAGNO(), USE

MDY()

Date and time data

Returns a specified date as a character string in MONTH DD, YY format.

Syntax

MDY(<expD>)

<expD> The date expression to return as a character string in MONTH DD, YY format.

Description

MDY() returns a date in MONTH DD, YY or MONTH DD, YYYY format, where MONTH is the full month name, DD is the day number, and YY is the year number. If SET CENTURY is OFF (the default), MDY() returns the year as 2 digits. If SET

CENTURY is ON, MDY() returns the year as 4 digits. MDY() always returns the day portion as 2 digits.

Enter *<expD>* in the current date format as determined by SET DATE, DBASEWIN.INI, or the International option of the Windows Control Panel (in that order). That is, SET DATE settings override those in DBASEWIN.INI, and settings in DBASEWIN.INI override those in the Windows Control Panel. Be sure *<expD>* matches the date format in use when your program runs.

If you pass an invalid date to MDY(), dBASE converts the date to a valid one and returns that date in MONTH DD, YY or MONTH DD, YYYY format. If you pass an empty or non-date expression delimited with braces ({ }) to MDY(), it returns "Unknown 00, 00" or "Unknown 00, 0000". If you pass a non-date expression or an expression that isn't delimited with braces to MDY(), it returns an error.

Example

See DMY() for an example of using MDY(), substituting MDY() for any DMY() reference.

Portability

Not supported in dBASE III PLUS.

See Also

CROW(), CMONTH(), DAY(), DMY(), DOW(), MONTH(), SET CENTURY, YEAR()

MEMLINES()

Fields and records

Returns the number of lines in a memo field.

Syntax

MEMLINES(*<memo field>* [*<line length expN>*])

<memo field> The memo field the MEMLINES() function operates on.

<line length expN> Specifies the line length used in calculating the number of lines in a memo field. *<expN>* can be set to any number from 8 to 255. If *<expN>* is not specified, MEMLINES() calculates each line using the memo width specified using the SET MEMOWIDTH command.

Description

The MEMLINES() function returns the number of lines in a memo field based on the memo width specified by the line length parameter. If you don't specify a line length, MEMLINES() treats the memo field text as if it were wordwrapped within a display—50 characters wide unless you specify another display width with the SET MEMOWIDTH command.

If a word doesn't completely fit within the remainder of a line, MEMLINES() includes that word at the beginning of the next line. If the number of characters in a word is longer than the default or specified memo field line length, MEMLINES() truncates the

word at the end of the line and includes the remainder of the word at the beginning of the next line.

Example

The following example uses MEMLINES() to return the number of lines in a memo field as a function of a user entered MEMOWIDTH. As a line counter variable is incremented down the page, MEMLINE() is used to determine whether the next memo will fit on the current page:

```
SET TALK OFF
CLEAR
colwidth = 10
@ 6,12 SAY "Report will be how many columns wide? " ;
  GET colwidth PICTURE "99" VALID colwidth > 9 .AND. colwidth < 68
READ
mwidth = SET("MEMOWIDTH")
SET MEMOWIDTH TO colwidth
USE Company
output_to = "S"
IF output_to = "S"
  pglngth = 15
ELSE
  pglngth = 58
ENDIF
linecnt = 1
memoline = 1
DO WHILE .NOT. EOF()
  CLEAR
  linecnt = 1
  DO WHILE linecnt + MEMLINES(Notes) <= pglngth .AND. .NOT. EOF()
    ? Company AT 2
    linecnt = linecnt + 1
    memoline = 1
    DO WHILE memoline <= MEMLINES(Notes) .AND. MEMLINES(Notes) <> 0
      ? MLINE(Notes,memoline)AT 12
      linecnt = linecnt + 1
      memoline = memoline + 1
    ENDDO
    SKIP
  ENDDO
  WAIT "Press any key to continue"
ENDDO
CLOSE ALL
SET MEMOWIDTH TO mwidth
```



Portability

Not supported in dBASE III PLUS. The line length argument is not supported in dBASE IV.

See Also

MLINE(), SET MEMOWIDTH, STORE MEMO

MEMORY()

Environment

Returns the number of kilobytes currently available in random access memory (RAM).

Syntax

MEMORY([<expN>])

<expN> *Visual* dBASE ignores this parameter; it is included for backward compatibility with dBASE IV.

Description

Use MEMORY() to determine how much memory is available in the system. The value MEMORY() returns includes memory made available by the use of a Windows swap file.

You might want to check available memory before using a command such as RUN or RUN() from within an application, to make sure the called application will load and run properly.

Example

The following example warns if the user has less than a megabyte of RAM available:

```
IF MEMORY()>=1000
  ? "You have at least a megabyte of RAM"
ELSE
  ?? "Warning: You have less than a megabyte of RAM:"
  ? MEMORY(),"Kb"
ENDIF
```

Portability

Not supported in dBASE III PLUS. In dBASE IV, <expN> can be a value from 0 to 7, with each value returning memory information relevant in a DOS environment.

See Also

DIR, DISPLAY MEMORY, FLUSH, LIST, RUN, RUN()

MENU()

dBASE IV menus

Returns the name of the current dBASE IV menu bar as an uppercase character string. This command is supported primarily for compatibility with dBASE IV. In *Visual* dBASE, use INSPECT() to return information associated with objects in forms.

For complete syntax information on MENU(), see online Help. For information about defining forms, see the Forms chapters in the *User's Guide*.

MESSAGE()

Error handling and debugging

Returns the error message of the most recent dBASE error.

Syntax

MESSAGE()

Description

Use MESSAGE() with other error-trapping commands and functions, such as ON ERROR, RETRY, and ERROR(), to substitute specific responses and actions for dBASE default responses to errors.

MESSAGE() is initially set to an empty string. MESSAGE() returns an error message when an error occurs, and remains set to that error message until one of the following happens:

- Another error occurs
- RETRY is issued
- The subroutine in which the error occurs completes execution

To learn the IDAPI error message of the last IDAPI error generated by the current table, use DBMESSAGE().

See the table in the description of ERROR() that compares CERROR(), ERROR(), MESSAGE(), DBERROR(), DBMESSAGE(), SQLERROR(), and SQLMESSAGE().

See online Help for a listing of all dBASE error messages.

M

Example

See ON ERROR for an example of using MESSAGE().

Portability

The error messages of some errors in dBASE IV and dBASE III PLUS are different from the error messages for the same errors in *Visual* dBASE. See online Help for more information.

See Also

CERROR(), DBERROR(), DBMESSAGE(), ERROR(), ON ERROR, RETRY, SQLERROR(), SQLMESSAGE()

MIN()

Expressions and type conversion

Compares two expressions of the same data type and returns the lesser value. If comparing two logical expressions, returns .T. if one or both expressions are true.

Syntax

MIN(<exp 1>, <exp 2>)

<exp 1> An expression (character, date, logical, float, or numeric) to compare to a second expression of the same data type. You can compare float to numeric type numbers.

<exp 2> An expression of the same data type as <exp 1> to compare to <exp 1>. You can compare float to numeric type numbers.

Description

Use MIN() to compare two numbers, two character strings, two dates, or two logical values to determine the lesser of the two values compared. MIN() returns:

- The smaller of two numbers
- The character string with the lower *collation value* of two character strings. Collation values are determined by the language driver in use. For more information on language drivers, see Appendix C in the *Programmer's Guide*.
- The earlier of two dates
- False if one or both of two logical expressions evaluates to false

If <exp 1> and <exp 2> are equal, MIN() returns their value. For example, if you issue STORE 100 TO mvar1 and STORE 50+50 TO mvar2, MIN(mvar1,mvar2) returns 100.

If you use MIN() with character strings, first issue SET EXACT ON to ensure accurate results. When comparing character strings, MIN() is case-sensitive.

Example

The following example uses MIN() to compare a YTD_Sales field value with a memory variable that holds a computed average and returns the smaller amount:

```
CLEAR
SET TALK OFF
USE Company
AVERAGE YTD_Sales FOR YTD_Sales<>0 TO Midpt
GO TOP
? CENTER("Subsidiaries Below Corporate Average Sales")
?
? "Company", "Subsid Avg" AT 27, "YTD Sales" AT 42, "Amount Below Avg" AT 56
? REPLICATE(" ",7) AT 0, REPLICATE(" ",10) AT 27, ;
  REPLICATE(" ",9) AT 42, REPLICATE(" ",16) AT 56
DO WHILE .NOT. EOF()
  IF MIN(YTD_Sales,MidPt)=Ytd_Sales
    ? Company, Midpt AT 25, YTD_Sales, Midpt - YTD_Sales
  ENDIF
  SKIP
ENDDO
RETURN
CLOSE ALL
```

Portability

In dBASE III PLUS, MIN() supports only numeric expressions. In dBASE IV, MIN() doesn't support logical expressions.

See Also

CALCULATE, IIF(), MAX(), SET EXACT

MKDIR**Disk and file utilities**

Creates a new DOS directory.

Syntax

MKDIR <directory>

<directory> The directory you want to create.

Description

Use MKDIR to create a new directory without exiting dBASE. MD is equivalent to MKDIR.

MKDIR is equivalent to the DOS command MKDIR or MD. When <directory> is a single name, MKDIR creates a new directory of that name beneath the current directory. When <directory> includes a path, MKDIR creates a new directory in the specified path. The path name must be an existing path, with the name of the new directory at the end preceded by a backslash. If <directory> is preceded by a backslash, MKDIR creates a new directory under the root directory.

The new directory name must follow the same naming conventions as DOS directory names.

After you create the new directory, you can use CD or SET DIRECTORY to make the new directory the default directory. You can also use SET PATH to include the new directory in the dBASE search path.

M**Example**

The following examples use MKDIR or MD to create a subdirectory on drive D called Project, three subdirectories below Project and another subdirectory on the C drive:

```
MKDIR D:\Project
MKDIR D:\Project\Programs
MKDIR D:\Project\Data
MD D:\Project\Backup
MD C:\Editor
```

If you try to make a directory that already exists or is on a path that does not exist you will get an error message

Portability

Not supported in dBASE IV or dBASE III PLUS.

See Also

CD, SET DIRECTORY, SET PATH

MLINE()

Extracts a specified line of text from a memo field in the current record.

Syntax

MLINE(<memo field> [, <line number expN> [, <line length expN>]])

<memo field> The memo field the MLINE() function operates on.

<line number expN> The number of the line in the memo field returned by the MLINE() function. The default for <line number expN> is 1.

<line length expN> The number that determines the length of a line in the memo field. <line length expN> can be set to any number from 8 to 255. If <line length expN> is not set, the SET MEMOWIDTH setting specifies the length of the line. If <line number expN> of the memo field has less than <line length expN> characters, MLINE() adds characters to the end of the returned string from the line following <line number expN>.

Description

MLINE() returns a specified line of text from a memo field. MLINE() treats the text of the memo field as if it were wordwrapped within a display width specified by the SET MEMOWIDTH setting or by <line length expN>. If a word doesn't completely fit within the specified length, MLINE() includes that word at the beginning of the next line and doesn't include any portion of the word in the returned string. Thus, the text that appears on a particular line is determined by the length of each line and wordwrapping text.

Example

See MEMMLINES() for an example of MLINE().

Portability

Not supported in dBASE III PLUS. The line length argument is not supported in dBASE IV.

See Also

MEMMLINES(), REPLACE MEMO, SET MEMOWIDTH, STORE MEMO

MOD()

Returns the modulus (remainder) of one number divided by another.

Syntax

MOD(<dividend expN>, <divisor expN>)

<dividend expN> The number to be divided.

<divisor expN> The number to divide by.

Description

MOD() divides <dividend expN> by <divisor expN> and returns the remainder as a whole number. For example MOD(X,Y) returns the remainder of x/y .

MOD() returns a positive number if <divisor expN> is positive, and returns a negative number if <divisor expN> is negative.

The modulus formula is

$$\text{<dividend>-FLOOR(<dividend>/<divisor>)*<divisor>}$$

where FLOOR() returns the greatest integer less than or equal to its argument.

Example

An example of MOD() is shown in the example for INT().

See Also

CEILING(), FLOOR(), INT()

MODIFY...

Information

Modifies the corresponding object or file.

Syntax

MODIFY APPLICATION
[<filename> | ? | <filename skeleton>]

MODIFY CATALOG
[<filename> | ? | <filename skeleton>]

MODIFY COMMAND
[<filename> | ? | <filename skeleton>]
[WINDOW <window name>]

MODIFY FILE
[<filename> | ? | <filename skeleton>]
[WINDOW <window name>]

MODIFY FORM
[<filename> | ? | <filename skeleton>]

MODIFY LABEL
[<filename> | ? | <filename skeleton>]

MODIFY MENU
[<filename> | ? | <filename skeleton>]

MODIFY POPUP
[<filename> | ? | <filename skeleton>]

M

Forms

Table basics

Programs

Disk and file utilities

Forms

Input/Output

Forms

Forms

MODIFY QUERY [<filename> ? <filename skeleton>] MODIFY REPORT [<filename> ? <filename skeleton>]	Table organization Input/Output
MODIFY SCREEN [<filename> ? <filename skeleton>]	Forms
MODIFY VIEW [<filename> ? <filename skeleton>]	Table organization

Description
The MODIFY commands listed here operate the same as their CREATE command counterparts. For more information, see the corresponding CREATE commands.

MODIFY STRUCTURE

Table basics

Allows you to modify the structure of the current table. You can modify the structure of both dBASE .DBF and Paradox .DB files, but not SQL tables.

Syntax
MODIFY STRUCTURE

Description
Use MODIFY STRUCTURE to change the structure of the current table by adding or deleting fields, or changing a field name, width, or data type. Issuing the MODIFY STRUCTURE command opens the Table Designer, an interactive environment in which you can create or modify the structure of a table. For more information about using the Table Designer, see the *User's Guide*.

Before allowing changes to the structure of a dBASE table, *Visual* dBASE makes a backup of the original table assigning the file a .DBK extension. *Visual* dBASE then creates a new table file with the .DBF extension and copies the modified table structure to that file. When you've finished modifying a table structure, *Visual* dBASE copies the content of the backup file into the new structure. If data is accidentally truncated or lost, you can recover the original data from the .DBK file. Before modifying the structure of a table, make sure that you have sufficient disk space to create the backup file plus any temporary storage required to copy records between the two tables (approximately twice the size of the original table).

If a table contains a memo field, MODIFY STRUCTURE also creates a backup memo file to store the original memo field data. This file has the same name as the table, but is given a .TBK extension.

You shouldn't change a field name and its width or type at the same time. If you do, *Visual* dBASE won't 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, and then use MODIFY STRUCTURE again to change the field width or data type.

Also, don't insert or delete fields from a table 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 change 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.

Visual dBASE successfully converts data between a number of field types. 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, *Visual* dBASE converts numbers from the numeric fields to right-aligned character strings. If you convert a character field to a numeric field, *Visual* dBASE converts 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, and vice versa. You can also convert character strings that 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. You can't convert logical fields to numeric fields.

In general, *Visual* dBASE attempts 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.

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. If any indexes are open when you modify a table structure, dBASE automatically closes those indexes when saving the modified table. You should re-index the table after you modify its structure.

Example

The following example uses MODIFY STRUCTURE in the Command window to change the structure of a table:

```
USE Clients IN SELECT() EXCLUSIVE
MODIFY STRUCTURE
CLOSE DATABASES
```

The structure of a table can also be displayed but not changed by using the following commands in the Command window:

```
USE CLIENTS IN SELECT() NOUPDATE
DISPLAY STRUCTURE
CLOSE DATABASES
```

See Also

APPEND, APPEND MEMO, COPY STRUCTURE, CREATE, DISPLAY STRUCTURE, LIST STRUCTURE, REPLACE

MONTH()

Returns the number of the month for a specified date expression.

Syntax

MONTH(<expD>)

<expD> The date expression whose corresponding month number to return.

Description

MONTH() returns a date's month number—a value from 1 to 12.

Enter <expD> in the current date format as determined by SET DATE, DBASEWIN.INI, or the International option of the Windows Control Panel (in that order). That is, SET DATE settings override those in DBASEWIN.INI, and settings in DBASEWIN.INI override those in the Windows Control Panel. Be sure <expD> matches the date format in use when your program runs.

If you pass an invalid date to MONTH(), dBASE converts the date to a valid one and returns the month number of that date. If you pass an empty or non-date expression delimited with braces ({ }) to MONTH(), it returns 0. If you pass a non-date expression or an expression that isn't delimited with braces to MONTH(), it returns an error.

Example

The following example uses MONTH() to determine the month number of the system date and combine it with a text string that includes the correct suffix:

```
IF SET("CENTURY") <> "OFF"
  SET CENTURY OFF
ENDIF
IF SET("DATE") <> "AMERICAN"
  SET DATE AMERICAN
ENDIF
month = MONTH( DATE() )
mon_strng = LTRIM( STR( month ) )
DO CASE
  CASE month = 1
    mon_strng = "1st"
  CASE month = 2
    mon_strng = "2nd"
  CASE month = 3
    mon_strng = "3rd"
  CASE month = 4 .OR. month = 5 .OR. month = 6 ;
    .OR. month = 6 .OR. month = 7 .OR. month = 8 ;
    .OR. month = 9 .OR. month = 10 ;
    .OR. month = 11 .OR. month = 12
    mon_strng = mon_strng + "th"
ENDCASE
? "This is the " + mon_strng + " month of " + ;
  LTRIM( STR( YEAR( DATE() ) ) ) + " " + DTOC( DATE() )
```

See Also

CMONTH(), DAY(), DATE(), SET CENTURY, SET DATE

MOVE WINDOW

dBASE IV windows

Redefines the row and column coordinates of the existing dBASE IV-style window *<window name>*. The window size, shape, and contents are not affected. This command is supported primarily for compatibility with dBASE IV. In *Visual dBASE*, use the Moveable property of a form to control whether the user can move the form.

For complete syntax information on MOVE WINDOW, see online Help. For information about defining forms, see the Forms chapters in the *User's Guide*.

MROW()

Keyboard and mouse events

Returns the current row position of the mouse pointer. This command is supported primarily for compatibility with dBASE IV. In *Visual dBASE*, use properties such as OnLeftMouseDown to manage mouse actions in forms.

For complete syntax information on MROW(), see online Help. For information about working with forms, see the Forms chapters in the *User's Guide*.

MSGBOX()

Forms

M

Opens a dialog box that displays a message and pushbuttons, and returns a numeric value that corresponds to the pushbutton the user chooses.

Syntax

MSGBOX(*<message expC>*, [*<title expC>*, [*<box type expN>*]])

<message expC> The message to display in the dialog box.

<title expC> The title to display in the title bar of the dialog box.

<box type expN> A numeric value that determines which icon (if any) and which pushbuttons to display in the dialog box. To specify a dialog box with pushbuttons and no icon, use the following numbers:

<box type expN>	Pushbuttons
0	OK
1	OK, Cancel
2	Abort, Retry, Ignore
3	Yes, No, Cancel
4	Yes, No
5	Retry, Cancel

MSGBOX()

For example, the following command opens a dialog box with Yes, No, and Cancel pushbuttons, and no icon.

```
Verdict = MSGBOX("Click me", "Bye!", 3)
```

To specify a dialog box with pushbuttons and an icon, add any of the following numbers to *<box type expN>*:

Number to add	Icon displayed
16	!
32	?
48	!
64	i

For example, the following command opens the same dialog box as the previous one, this time with a blue i:

```
? MSGBOX("Click me", "Bye!", 67) && 3 + 64
```

When a dialog box has more than one pushbutton, the left most pushbutton is normally the default. However, if you add 258 to *<box type expN>*, the second pushbutton is the default, and if you add 512 to *<box type expN>*;, the third pushbutton is the default. For example, the following command opens the same dialog box as the previous one with the Cancel pushbutton as the default:

```
? MSGBOX("Click me", "Bye!", 579) && 3 + 64 + 512
```

If you omit *<box type expN>*, box type 0 is used by default.

Description

Use MSGBOX() to prompt the user to make a choice or acknowledge a message by clicking a pushbutton in a modal dialog box.

While the dialog box is open, program execution stops and the user cannot give focus to another window. When the user chooses a pushbutton, the dialog box disappears, program execution resumes, and MSGBOX() returns a numeric value that indicates which pushbutton was chosen.

Pushbutton	Return value
OK	1
Cancel	2
Abort	3
Retry	4
Ignore	5
Yes	6
No	7

For example, you can use the return value to make branching decisions in an error handling routine. Each time an error condition occurs, the routine could use MSGBOX() to open a dialog box containing Abort, Retry, and Ignore pushbuttons. The routine could evaluate the number returned by MSGBOX() and take the appropriate action.

Example

The following example uses an error handling routine to display a dialog box offering Abort, Retry, and Ignore pushbuttons each time an error condition is generated. If the user clicks Abort, execution is terminated. If the user clicks Retry, control returns to the master routine. If the user clicks Ignore, execution continues after the command that failed.

```
* The master routine
SET TALK OFF
FOR i = 1 to 5
    DO MessUp
        ? "Try again" && This command executes only if
                        && user clicks the Retry button.
NEXT i
RETURN

PROCEDURE MessUp
    ON ERROR DO FixIt WITH;
        PROGRAM(), MESSAGE(), LINENO()

    ? COMPANY()      && This function does not exist,
                        && so it generates an error.
    DIR               && This command executes only if
                        && user clicks Ignore button.
RETURN

PROCEDURE FixIt(PRO, MES, LIN)
    Verdict = MSGBOX("Problem at line "+;
        LTRIM(STR(LIN))+;
        ", "+MES, "Routine's name: "+PRO, 18)
    DO CASE
        CASE Verdict = 3 && Abort (Stop execution)
            CANCEL
        CASE Verdict = 4 && Retry (Try again)
```

NDX()

```
        RETURN TO MASTER
CASE Verdict = 5  && Ignore (Go on)
    RETURN
ENDCASE
```

RETURN **Portability**

Not supported in dBASE IV or dBASE III PLUS.

See Also

ACCEPT, READMODAL(), ReadModal(), WAIT

NDX()

Table organization

Returns the name of an .NDX file.

Syntax

NDX([<index position expN> [, <alias>]])

<index position expN> Selects an .NDX file by the position of an index file in the list of open indexes for the current or a specified table.

<alias> A work area number (1 through 225), letter (A through J), or alias name. The work area letter or alias name must be enclosed in quotes.

Description

The NDX() function returns the name of an .NDX file based on the position of an index file in the list of open indexes for the current or a specified table. The position of indexes in the index file list is specified by SET INDEX, SET ORDER, or USE commands.

If you do not specify an index position, NDX() returns the name of the current master index file, or an empty string ("") if the master index is provided by an .MDX file. NDX() also returns an empty string if there is no index in the specified index position for the list of open indexes defined by the SET INDEX or USE commands.

If SET FULLPATH is ON, NDX() also returns the drive and directory location of the .NDX file in addition to its name.

Example

The following example creates three indexes and then uses NDX() to determine the names of .NDX files open for the current table:

```
USE Company EXCLUSIVE
INDEX ON CompCode TO CompCode
INDEX ON City TO City
INDEX ON Zip_P_code TO Zip
SET INDEX TO Zip, CompCode, City
* Now all 3 indexes are open
? "Index 1 ",NDX(1)
? "Index 2 ",NDX(2)
? "Index 3 ",NDX(3)
```

See Also

DBF(), FIELD(), KEY(), MDX(), ORDER(), SET FULLPATH, SET INDEX, SET ORDER, TAG(), USE

NETWORK()**Shared data**

Returns .T. if dBASE is running on a system in which a *local area network* (LAN) card or other multiuser system card has been installed.

Syntax

NETWORK()

Description

Use NETWORK() to determine if a program might be running in a network environment. For example, your program might need to do something in a network environment that it doesn't need to do in a single-user environment, such as issue USE with the EXCLUSIVE option.

NETWORK() returns .T. if a network card is installed; it doesn't determine whether a user is currently running dBASE in a network environment. To determine whether a user is actually working in a network environment, use ID().

Example

This example uses NETWORK() to test if the user has a network card installed. The user will SET EXCLUSIVE ON only when it might be needed. Without a network card, SET EXCLUSIVE can be ON permanently:

```
IF NETWORK( )
    SET EXCLUSIVE OFF && set ON as needed
ELSE
    SET EXCLUSIVE ON && No network
ENDIF
```

See Also

GETENV(), OS(), USE

N**NEXTKEY()****Keyboard and mouse events**

Returns the decimal value associated with a key or key combination held in the keyboard typeahead buffer. NEXTKEY() does not remove the keystroke from the buffer.

Syntax

NEXTKEY([<expN>])

NEXTKEY()

<expN> The position of the key or key combination in the typeahead buffer. If **<expN>** is omitted, NEXTKEY() returns the value of the first keystroke in the buffer. If **<expN>** is larger than the number of keystrokes in the buffer, NEXTKEY() returns 0.

Description

The keyboard typeahead buffer stores keystrokes the user enters while dBASE is busy processing other data. Use NEXTKEY() to identify, but not delete, a keystroke held in the buffer. If the buffer is empty, NEXTKEY() returns a value of zero.

For example, if you press *C* and then *Alt+P*, dBASE stores the values 67 and -420 in the typeahead buffer. NEXTKEY(1) returns 67, and NEXTKEY(2) returns -420. See Appendix D for a complete list of the values returned by NEXTKEY(). See Appendix E for a complete list of decimal values corresponding to keys.

To determine the value of the first keystroke in the buffer and delete it from the buffer, use INKEY().

Example

The following example verifies whether SPACEBAR is the key pressed immediately after F6. If so, the SpaceBar routine is called; otherwise, the typeahead buffer is cleared:

```
WAIT
IF LASTKEY() = -5      && F6
  IF NEXTKEY() = 32    && SPACEBAR
    DO SpaceBar
  ELSE
    CLEAR TYPEAHEAD
```

Description

OEM() is the inverse of ANSI(). For more information, see ANSI().

Example

The following example displays the 255 characters that are possible with ASCII, ANSI, and OEM formats:

```
FOR i=1 to 255
  ASCII=CHR(i)
  ? i,ASCII,ANSI(ASCII),OEM(ASCII)
  * The next 3 commands cause a pause
  * every 10 lines
  IF MOD(i,10)=0
    WAIT
  ENDF
NEXT i
```

Portability

Not supported in dBASE IV or dBASE III PLUS.

See Also

ANSI()

ON BAR

dBASE IV menus

Executes a command when the user selects (highlights) a bar in a dBASE IV popup menu. This command is supported primarily for compatibility with dBASE IV. In *Visual dBASE*, use `DEFINE`, `OPEN FORM`, and `READMODAL()` to create and activate menus associated with forms.

For complete syntax information on `ON BAR`, see online Help. For information about defining forms, see the Forms chapters in the *User's Guide*.

ON ERROR

Error handling and debugging

Executes a specified command when an error occurs.

Syntax

`ON ERROR`
[*<command>*]

<command> The command to execute when an error occurs in the program, procedure, or UDF. To execute more than one command when an error occurs, issue `ON ERROR DO <filename>`, where *<filename>* is a program or procedure file containing the sequence of commands to execute. `ON ERROR` without a *<command>* option disables any previous `ON ERROR <command>`.

Description

Use `ON ERROR` to control a program's response to *run-time* errors. For example, *<command>* can specify a new message in place of a standard error message. When `ON ERROR` is active, dBASE doesn't display its default run-time error messages.

While dBASE is executing an `ON ERROR` command, that particular `ON ERROR <command>` statement is disabled. Thus, if another error occurs during the execution of *<command>*, dBASE responds with its default error messages. You can, however, set another `ON ERROR` condition inside a subroutine called with `ON ERROR`.

`ON ERROR <command>` has no effect during the following commands:

- `APPEND`
- `CREATE/MODIFY`
 `APPLICATION/LABEL/REPORT/SCREEN/STRUCTURE/VIEW`
- `MODIFY COMMAND/FILE`

`ON ERROR` is similar to `ON ESCAPE` and `ON KEY`. `ON ESCAPE` specifies a command for dBASE to execute when `SET ESCAPE` is `ON` and *Esc* is pressed. `ON KEY` specifies a command for dBASE to execute when a specified key is pressed.

Avoid using a dBASE command recursively with `ON ERROR`.

See the table in the description of `ERROR()` that compares `CERROR()`, `ERROR()`, `MESSAGE()`, `DBERROR()`, `DBMESSAGE()`, `SQLERROR()`, and `SQLMESSAGE()`.

Example

The following example uses ON ERROR to open a user defined window when a run-time error occurs. In the example, the command ? Company() will cause an error because Company function does not exist:

```
ON ERROR DO ErrHndlr WITH ERROR(), MESSAGE(), PROGRAM(), LINENO()
USE Clients
? Company()
RETURN

PROCEDURE ErrHndlr
PARAMETERS nErrorNo, cErrorMessage, cProgram, nLineNo
DEFINE FORM HeadsUp FROM 10,25 TO 20,50
DEFINE TEXT Line1 OF HeadsUp AT 2,2 ;
    PROPERTY Text "An Error has occurred", Width 22
DEFINE TEXT Line2 OF HeadsUp AT 4,2;
    PROPERTY Text "Error: " + cErrorMessage, Width 22
DEFINE TEXT Line3 OF HeadsUp AT 5,2;
    PROPERTY Text "Number: " + STR(nErrorNo), Width 22
DEFINE TEXT Line4 OF HeadsUp AT 6,2;
    PROPERTY Text "Program: "+ cProgram, Width 22
DEFINE TEXT Line5 OF HeadsUp AT 7,2;
    PROPERTY Text "Line #: " + STR(nLineNo), Width 22
OPEN FORM HeadsUp
```

See Also

CERROR(), ERROR(), DBERROR(), DBMESSAGE(), MESSAGE(), ON ESCAPE, ON KEY, ON READERROR, RETRY, RETURN, SET ERROR, SET ESCAPE, SQLERROR(), SQLMESSAGE()

ON ESCAPE

Keyboard and mouse events

Executes a specified command when SET ESCAPE is ON and the *Esc* key is pressed during command or program execution.

Syntax

```
ON ESCAPE
[<command>]
```

<command> The command to execute when the following conditions are in effect:

- SET ESCAPE is ON
- The user presses *Esc* during command or program execution

To execute more than one command when *Esc* is pressed, issue ON ESCAPE DO <filename>, where <filename> is a program or procedure file containing the sequence of commands to execute. ON ESCAPE without a <command> option disables any previous ON ESCAPE <command>.

If you issue ON ESCAPE <command> in a program, you should disable the current ON ESCAPE condition by issuing ON ESCAPE without a <command> option before the

program ends. Otherwise, the ON ESCAPE condition remains in effect for any subsequent commands and programs you issue and run until you exit dBASE.

Description

When SET ESCAPE is ON (its default setting), pressing *Esc* interrupts program execution. If ON ESCAPE *<command>* is in effect, pressing *Esc* executes the specified command and then continues program execution. If no ON ESCAPE *<command>* is in effect, pressing *Esc* interrupts program execution and displays the dBASE Program Interrupted window.

You can issue ON ESCAPE *<command>* in the Command window to execute *<command>* when *Esc* is pressed during the execution of subsequent commands. In this case, dBASE returns control to the Command window when it finishes executing *<command>*.

Pressing *Esc* while a user-defined window, menu bar, pop-up menu, or pull-down menu is active deactivates the object and *doesn't* execute the command specified with ON ESCAPE. ON ESCAPE *<command>* also has no effect when *Esc* is pressed during commands such as CHANGE, EDIT, or READ.

ON ESCAPE is similar to ON ERROR and ON KEY. ON ERROR specifies a command for dBASE to execute when an error occurs in a program, procedure, or UDF. ON KEY specifies a command for dBASE to execute when a specified key, or any key, is pressed.

If SET ESCAPE is ON and both ON KEY and ON ESCAPE are in effect, ON ESCAPE takes precedence when *Esc* is pressed. See ON KEY for a table that summarizes the relationship between SET ESCAPE, ON ESCAPE, and ON KEY when *Esc* is pressed.

Example

The following example uses ON ESCAPE to substitute a small program, PrgEscape, for the Escape key. In this example, the programmer has wrongly programmed an endless loop that requires the use of Escape to debug. When escape is pressed, the program and line will be shown:

```
SET PROCEDURE TO PROGRAM(1) ADDITIVE
* So that other program files can access PrgEscape
SET ESCAPE ON
ON ESCAPE DO PrgEscape WITH PROGRAM(1),LINENO()
DO WHILE .t.  && set up an endless loop
  x="always allow a way out"
  y=" of a do while loop"
  z="Press Escape to stop"
  ? x
  ? y
  ? z
ENDDO

Procedure PrgEscape
PARAMETERS prg,line
? "Programmed Escape:",prg,line
SUSPEND
```

See Also

ON ERROR, ON KEY, SET ESCAPE, SET KEY

ON EXIT BAR

dBASE IV menus

Executes a command when the user exits a pop-up menu item. This command is supported primarily for compatibility with dBASE IV. In *Visual dBASE*, use DEFINE, OPEN FORM, and READMODAL() to create and activate menus associated with forms.

For complete syntax information on ON EXIT BAR, see online Help. For information about defining forms, see the Forms chapters in the *User's Guide*.

ON EXIT MENU

dBASE IV menus

Executes a command when the user exits a pad in a dBASE IV menu bar if the pad is not assigned a command by ON EXIT PAD. This command is supported primarily for compatibility with dBASE IV. In *Visual dBASE*, use DEFINE, OPEN FORM, and READMODAL() to create and activate menus associated with forms.

For complete syntax information on ON EXIT MENU, see online Help. For information about defining forms, see the Forms chapters in the *User's Guide*.

ON EXIT PAD

dBASE IV menus

Executes a command when the user exits a specified pad in a dBASE IV menu bar. This command is supported primarily for compatibility with dBASE IV. In *Visual dBASE*, use DEFINE, OPEN FORM, and READMODAL() to create and activate menus associated with forms.

For complete syntax information on ON EXIT PAD, see online Help. For information about defining forms, see the Forms chapters in the *User's Guide*.

ON EXIT POPUP

dBASE IV menus

Executes a command when the user exits a bar in a dBASE IV pop-up menu, if the bar is not assigned a command by ON EXIT BAR. This command is supported primarily for compatibility with dBASE IV. In *Visual dBASE*, use DEFINE, OPEN FORM, and READMODAL() to create and activate menus associated with forms.

For complete syntax information on ON EXIT POPUP, see online Help. For information about defining forms, see the Forms chapters in the *User's Guide*.

ON KEY

Keyboard and mouse events

Executes a command when a specified key or key combination is pressed during execution of a command or a program.

Syntax

```
ON KEY
[LABEL <key label>]
[<command>]
```

LABEL <key label> Identifies the key or key combination that, when pressed, causes <command> to execute. Without LABEL <key label>, dBASE executes <command> when you press any key. ON KEY LABEL is not case-sensitive.

<command> The command that is executed when you press the key or key combination. If you omit <command>, the command previously assigned by ON KEY is disabled.

Description

Use ON KEY to specify a command that executes when the user presses a key or key combination. When the key is pressed, ON KEY causes dBASE to stop executing the current command and execute <command>. The interrupted command resumes when <command> finishes executing.

ON KEY interrupts most commands, but it doesn't interrupt commands that rely on uninterrupted processing to ensure data accuracy, such as SORT, INDEX, and PACK. ON KEY also doesn't affect the execution of some commands that require user input, such as ACCEPT and WAIT.

When you issue both ON KEY LABEL <key label> <command> and ON KEY <command>, the key or key combination you specify with ON KEY LABEL <key label> <command> takes precedence and executes its associated <command>.

When you don't include LABEL, only one ON KEY specification can be active at a given time. When you include LABEL, multiple ON KEY specifications for multiple keys can be active at once.

ON KEY without arguments removes the effect of all previously-entered ON KEY <command> commands.

SET KEY is identical in function to ON KEY LABEL; only the syntax differs. If you use ON KEY LABEL and SET KEY to set the same key, dBASE executes the program or procedure specified by the most recently issued command.

ON KEY is similar to ON ESCAPE. ON ESCAPE specifies a command for dBASE to execute when SET ESCAPE is ON and the *Esc* key is pressed. If SET ESCAPE is ON and both ON KEY and ON ESCAPE are in effect, ON ESCAPE takes precedence when *Esc* is pressed. The following table summarizes the relationship between SET ESCAPE, ON ESCAPE, and ON KEY when *Esc* is pressed:

SET ESCAPE	ON ESCAPE	ON KEY	When you press Esc
ON		DO prog1	dBASE displays Program Interrupted window
OFF		DO prog1	Prog1 executes
ON	DO prog2	DO prog1	Prog2 executes
OFF	DO prog2	DO prog1	Prog1 executes

ON KEY LABEL

Use the following key label names to assign key or key combinations with ON KEY LABEL <key label>.

Key identification	<key label>
Alphabetic characters	A or a, B or b, ...
Numbers	0, 1, 2 ...
Backspace	<i>Backspace</i>
Back Tab	<i>Backtab</i>
Delete	<i>Del</i>
End	<i>End</i>
Home	<i>Home</i>
Insert	<i>Ins</i>
Page Up	<i>PgUp</i>
Page Down	<i>PgDn</i>
Tab	<i>Tab</i>
Left arrow	<i>Leftarrow</i>
Right arrow	<i>Rightarrow</i>
Up arrow	<i>Uparrow</i>
Down arrow	<i>Downarrow</i>
F1 to F10	<i>F1, F2, F3, ...</i>
Control+<key>	<i>Ctrl-<key> or Ctrl+<key></i>
Shift+<key>	<i>Shift-<key> or Shift+<key></i>
Alt+<key>	<i>Alt-<key> or Alt+<key></i>
Enter	<i>Enter</i>
Escape	<i>Esc</i>
Space bar	<i>Spacebar</i>

Example

The following example displays selected fields from 10 records, pauses for 3 seconds and adds more records to the screen. ON KEY LABEL command is used to branch the program to procedures that either reverse the record pointer, enter a browse window, or exit scrolling:

```
CLEAR
SET TALK OFF
PUBLIC mExit
mExit=.F.
```

```

ON KEY LABEL F8 DO BackUp
ON KEY LABEL F9 DO Details
ON KEY LABEL F10 DO GetOut
USE Clients Order Company
? CENTER("INSTRUCTIONS")
? CENTER("Press F8 to go back X records")
? CENTER("Press F9 to Browse all fields")
? CENTER("Press F10 to exit")
?
WAIT "Slow Browse - Press any key when ready"
Cnt=1
DO WHILE .NOT. EOF() .AND. .NOT. mExit
    DISPLAY NEXT 10 Company, Contact
    Pause=INKEY(3)
ENDDO
SET TALK ON
ON KEY
RETURN

PROCEDURE BackUp
CLEAR
SKIP -9
RETURN

PROCEDURE Details
SKIP -9
BROWSE
RETURN

PROCEDURE GetOut
mExit=.T.
RETURN

```

Portability

The LABEL <key label> option is not supported in dBASE III PLUS.

See Also

CLEAR TYPEAHEAD, INKEY(), KEYBOARD, ON ERROR, ON ESCAPE, SET ESCAPE, SET FUNCTION

ON MENU

dBASE IV menus

Executes a command when the user selects (highlights) any pad in a dBASE IV menu bar if the pad is not assigned a command with ON PAD. This command is supported primarily for compatibility with dBASE IV. In *Visual* dBASE, use DEFINE, OPEN FORM, and READMODAL() to create and activate menus associated with forms.

For complete syntax information on ON MENU, see online Help. For information about defining forms, see the Forms chapters in the *User's Guide*.

ON MOUSE

Keyboard and mouse events

Detects when the user clicks the left mouse button and executes a command when the button is released. This command is supported primarily for compatibility with dBASE IV. In *Visual* dBASE, use properties such as `OnLeftMouseDown` to manage mouse actions in forms.

For complete syntax information on ON MOUSE, see online Help. For information about working with forms, see the Forms chapters in the *User's Guide*.

ON NETERROR

Shared data

Executes a specified command when a multiuser-specific error occurs.

Syntax

ON NETERROR [*<command>*]

<command> The command to execute when a multiuser-specific error occurs. To execute more than one command when such an error occurs, issue ON NETERROR DO *<filename>*, where *<filename>* is a program or procedure file containing the sequence of commands to execute. ON NETERROR without a *<command>* option disables any previous ON NETERROR *<command>* statement.

Description

Use ON NETERROR to control a program's response to multiuser-specific errors. For example, in a multiuser environment on a *local area network* (LAN), an error can occur when two users attempt to alter the same record in a shared table at the same time, or when one user attempts to open a shared table that another user already has open for exclusive use.

ON NETERROR is similar to ON ERROR, except that ON ERROR responds to all run-time errors regardless of whether they're multiuser-specific. You can use ON ERROR to handle both single-user and multiuser errors, or you can use ON NETERROR to handle just multiuser errors. If you issue both ON ERROR and ON NETERROR, then ON ERROR responds to just single-user errors, leaving ON NETERROR to respond to multiuser errors.

While dBASE is executing an ON NETERROR command, that particular ON NETERROR *<command>* statement is disabled. Thus, if another multiuser-specific error occurs during the execution of *<command>*, dBASE responds with its default error messages. You can, however, set another ON NETERROR condition inside a subroutine called with ON NETERROR.

You should avoid using a dBASE command recursively with ON NETERROR.

Example

The following example uses ON NETERROR in case the Clients table cannot be opened exclusively:


```

SET PROCEDURE TO PROGRAM(1) ADDITIVE
ON NETERROR Do NetErr
USE Clients EXCLUSIVE
* If Clients cannot be opened exclusively then
* the subroutine NetErr will be called.

```

```

PROCEDURE NETERR
WAIT "Multi-user problem"

```

See Also

DO, ERROR(), MESSAGE(), ON ERROR, RETRY, RETURN, SET EXCLUSIVE, SET REPROCESS, SQLERROR(), SQLMESSAGE()

ON PAD

dBASE IV menus

Executes a command or displays a pop-up menu when the user selects (highlights) a pad in a dBASE IV menu bar. This command is supported primarily for compatibility with dBASE IV. In *Visual dBASE*, use DEFINE, OPEN FORM, and READMODAL() to create and activate menus associated with forms.

For complete syntax information on ON PAD, see online Help. For information about defining forms, see the Forms chapters in the *User's Guide*.

ON PAGE

Printing

Executes a specified command when printed output reaches a specified line on the current page.

Syntax

```

ON PAGE
[AT LINE <expN> <command>]

```

AT LINE <expN> Identifies the line number at which to execute the specified page-formatting command.

<command> The command to execute when printed output reaches the specified line number, <expN>. To execute more than one command, issue ON PAGE DO <filename>, where <filename> is a program or procedure file containing the sequence of commands to execute.

Description

Use ON PAGE to specify a command to execute when printed output reaches a specific line number. ON PAGE with no options disables any previous ON PAGE statement.

The value of the _plineno system variable indicates the number of lines that have been printed on the current page. As soon as the _plineno value is equal to the value you specify for <expN>, dBASE executes the ON PAGE command.

Use the ON PAGE command to print *headers* and *footers*. For example, the ON PAGE command can call a procedure when the `_plineno` system memory variable reaches the line number that signifies the end of a page. In turn, that procedure can call two procedures, one to print the footer on the current page and one to print the header on the next page.

You can begin header routines with EJECT PAGE to ensure that the header text prints at the top of the following page. EJECT PAGE also sets the `_plineno` system memory variable to 0. Use the `?` command at the beginning of a header procedure to skip several lines before printing the header information. You can also use the `?` command at the end of the procedure to skip several lines before printing the text for the page.

Begin footer routines with the `?` command to move several lines below the last line of text. You can use the `??` command with the `_pageno` system memory variable to print a page number for each page on the same line as the footer.

To calculate the appropriate footer position, add the number of lines for the bottom margin and the number of lines for the footer text to get the total lines for the bottom of the page. Subtract this total from the total number of lines per page. Use this result to specify a number for the AT LINE argument. If the footer text exceeds the number of lines per page, the remainder prints on the next page.

Example

This example uses EJECT PAGE in conjunction with ON PAGE to print a footer on each page. In this example, a page length of 5 is set up (lines 0 through 4). Text is printed on four lines: 0,1,2,3, and the footer prints on line 4. Eight lines of text are printed on two pages:

```
SET TALK OFF
CLEAR
SET PRINTER ON
_padvance="LINEFEEDS"
_plength=5
_pageno=1
EJECT
ON PAGE AT LINE 3 do Page_Brk
PRINTJOB
i=1
DO WHILE i<=8
  ?? "Line of text  ",_pageno,_plineno,i
  ?
  i=i+1
ENDDO
ON PAGE  && turn off ON PAGE
ENDPRINTJOB
CLOSE PRINTER

PROCEDURE Page_Brk
?? "  Page Footer",_pageno,_plineno
EJECT PAGE
RETURN
*
*          Page          Line          i
* This prints out as:
```

*			
* Line of text	1	0	1
* Line of text	1	1	2
* Line of text	1	2	3
* Line of text	1	3	4
* Page Footing	1	4	
* Line of text	2	0	5
* Line of text	2	1	6
* Line of text	2	2	7
* Line of text	2	3	8
* Page Footer	2	4	

Portability

Not supported in dBASE III PLUS.

See Also

?, ??, _pageno, _plinenno, EJECT PAGE, PRINTJOB...ENDPRINTJOB, SET PCOL, SET PRINTER, SET PROW

ON POPUP

dBASE IV menus

Executes a command when the user selects (highlights) a bar in a dBASE IV pop-up menu, if the bar is not assigned a command with ON BAR. This command is supported primarily for compatibility with dBASE IV. In *Visual dBASE*, use DEFINE, OPEN FORM, and READMODAL() to create and activate menus associated with forms.

For complete syntax information on ON POPUP, see online Help. For information about defining forms, see the Forms chapters in the *User's Guide*.

ON READERROR

Error handling and debugging



Executes a specified command when an error occurs during data-entry.

Syntax

ON READERROR
[<command>]

<command> The command to execute when a user enters an invalid value into a field. ON READERROR without <command> disables any previously issued ON READERROR <command>.

Description

Use ON READERROR to trap the following data-entry errors:

- Invalid dates
- Input that is outside the boundaries specified by a RANGE option of @...SAY...GET
- Input that doesn't meet the condition specified in a VALID option of @...SAY...GET

The command you specify with ON READERROR can be a call to a program, procedure, or user-defined function (UDF). For example, you could call a UDF that displays a message.

With or without ON READERROR, the user must enter valid dates and data that are within a VALID or RANGE clause. If the user makes a data-entry error when a RANGE clause is in effect but no ON READERROR condition is in effect, dBASE displays a message indicating acceptable input values.

Example

The following example brings up the Clients table in a Table Record window and uses ON READERROR to trap for incorrect date entries. To activate ON READERROR, enter an invalid date in BalDate and procedure ErrHndlr presents a window with appropriate messages:

```
ON READERROR DO ErrHndlr
USE Clients
BROWSE
RETURN

PROCEDURE ErrHndlr
DEFINE FORM HeadsUp FROM 10,25 TO 20,50
DEFINE TEXT Line1 OF HeadsUp AT 2,4;
    PROPERTY Text "An Error has occurred",;
    Width 22, ColorNormal "R/W"
DEFINE TEXT Line2 OF HeadsUp AT 4,4;
    PROPERTY Text "Error: Date out of range",;
    Width 22
DEFINE TEXT Line3 OF HeadsUp AT 6,4;
    PROPERTY Text "Enter a value within range",;
    Width 22
OPEN FORM HeadsUp
```

Portability

Not supported in dBASE III PLUS.

See Also

APPEND, CHANGE, EDIT, INSERT, ON ERROR, READ, @...SAY...GET

ON SELECTION BAR

dBASE IV menus

Executes a command when the user chooses a bar in a dBASE IV pop-up menu. This command is supported primarily for compatibility with dBASE IV. In *Visual dBASE*, use DEFINE, OPEN FORM, and READMODAL() to create and activate menus associated with forms.

For complete syntax information on ON SELECTION BAR, see online Help. For information about defining forms, see the Forms chapters in the *User's Guide*.

ON SELECTION FORM

Executes a subroutine or a codeblock when the user submits a form.

Syntax

ON SELECTION FORM *<form name>*

$$[\langle expFP \rangle \mid \langle expCB \rangle]$$

<form name> The name of the form to which <expFP> or <expCB> is assigned.

<expFP> | <expCB> A subroutine or a codeblock to execute when the user submits the form. <expFP> is the name of a function or a procedure, and <expCB> is a codeblock.

If you omit *<expFP>* | *<expCB>*, the subroutine or codeblock previously assigned by ON SELECTION FORM is disabled.

Description

Use ON SELECTION FORM to execute a subroutine or codeblock automatically when the user submits a form.

A form is submitted when the user:

- Presses *Enter* when the cursor is in the body of the form, but not in a browse object, an editor object, or on a drive letter or directory prompt in a list box or a combo box.
- Presses *Spacebar* when the cursor is on a pushbutton.
- Presses the pick character of a pushbutton, or *Alt+<pick character>*, when the cursor is in the form.
- Clicks when the mouse pointer is on a pushbutton.

ON SELECTION FORM is equivalent to the OnSelection event property of the form object.

Example

The following example defines a form and list box for selecting an aircraft model and uses ON SELECTION FORM to call procedure Photo when the user presses Enter. Procedure Photo displays a graphic from the binary field Image of the selected record:

```

CLOSE ALL
SET TALK OFF
SET PROCEDURE TO PROGRAM(1) ADDITIVE
USE Aircrdb ORDER Aircraft IN SELECT()
SELECT Aircrdb
DEFINE FORM AC ;
    PROPERTY ;
        Top 5, ;
        Left 2, ;
        Height 13, ;
        Width 30, ;
        Text "Aircraft", ;
        Sizeable .T.
DEFINE LISTBOX Model OF AC ;

```

ON SELECTION MENU

```
PROPERTY                                ;
    Top 2,                              ;
    Left 10,                            ;
    Height 7,                           ;
    Width 18,                           ;
    DataSource "FIELD Aircrdb->Aircraft"
ON SELECTION FORM AC Photo
OPEN FORM AC

FUNCTION Photo
RESTORE IMAGE FROM BINARY Image
RETURN .T.
```

Portability

Not supported in dBASE IV or dBASE III PLUS.

See Also

ActiveControl, _curobj, OnClick, OnSelection

ON SELECTION MENU

dBASE IV menus

Executes a command when the user chooses a pad from a dBASE IV menu bar if the pad is not already assigned a command with ON SELECTION PAD. This command is supported primarily for compatibility with dBASE IV. In *Visual* dBASE, use DEFINE, OPEN FORM, and READMODAL() to create and activate menus associated with forms.

For complete syntax information on ON SELECTION MENU, see online Help. For information about defining forms, see the Forms chapters in the *User's Guide*.

ON SELECTION PAD

dBASE IV menus

Specifies the command that executes when the user chooses a pad in a dBASE IV menu bar. This command is supported primarily for compatibility with dBASE IV. In *Visual* dBASE, use DEFINE, OPEN FORM, and READMODAL() to create and activate menus associated with forms.

For complete syntax information on ON SELECTION PAD, see online Help. For information about defining forms, see the Forms chapters in the *User's Guide*.

ON SELECTION POPUP

dBASE IV menus

Executes a command when the user chooses a bar in a dBASE IV pop-up menu, if the bar is not assigned a command by ON SELECTION BAR. This command is supported primarily for compatibility with dBASE IV. In *Visual* dBASE, use DEFINE, OPEN FORM, and READMODAL() to create and activate menus associated with forms.

For complete syntax information on ON SELECTION POPUP, see online Help. For information about defining forms, see the Forms chapters in the *User's Guide*.

OPEN DATABASE

Table basics

Establishes a connection to a database server or a database defined for a specific directory location.

Syntax

```
OPEN DATABASE <database name>
[AUTOEXTERN]
[LOGIN <username>/<password>]
[WITH <option string>]
```

<database name> The name of the database you want to open. Databases are created using the BDE Configuration Utility (see *Getting Started* for more information).

<user name>/<password> Character string specifying the user name and password combination required to access the database.

WITH <option string> Character string specifying server-specific information required to establish a database server connection. For information about establishing database server connections, refer to your Borland SQL Link documentation, and contact your network or database administrator for specific connection information.

Description

The OPEN DATABASE command is used to establish a connection with a database defined with the BDE Configuration Utility. When opening a database, you need to specify whatever login parameters and database-specific information that connection requires. Typically, your network or system administrator can provide you with the information necessary to establish connections to established databases and database servers at your site.

Example

The following example uses OPEN DATABASE to establish a connection with a database server, opens a database previously created using the BDE Configuration Utility with SET DATABASE TO, and opens a server dBASE table and appends it to the client/server database:

```
CLEAR
SET DBTYPE TO DBASE
OPEN DATABASE CAClients      && Establish connection with database server
USE CAClients IN 1          && Opens server table
SELECT 1                    && Work area 1 active
APPEND FROM CLIENTS.DBF     && Append from local dBASE table
CLOSE ALL                   && Closes Clients server table
CLOSE DATABASE CAClients    && Disconnect from database server
```

Portability

Not supported in dBASE IV or dBASE III PLUS.

See Also

CLOSE..., DATABASE(), SET DATABASE, SET DBTYPE

OPEN FORM

Forms

Opens forms as modeless windows.

Syntax

```
OPEN FORM <form name 1> [ON <object name 1>]
[, <form name 2> [ON <object name 2>] ...]
```

<form name 1> [, **<form name 2>**, ...] The names of the forms to open.

ON <object name> Specifies which object in a form gets initial focus.

Description

Use OPEN FORM to open forms and display their objects.

The form you open with OPEN FORM is modeless. A modeless form has the following characteristics:

- While the form is open, focus can be transferred to other forms.
- Execution of the routine that opened the form continues after the form is opened.

You open forms as modeless windows when you want more than one form open at once. For example, an application that performs several tasks might use a different form for each task.

Only one form can have focus at a given time; this form is said to be *active*, and is displayed on top of all other open forms. dBASE gives input focus to forms in the order in which you list them in the OPEN FORM command; the first form specified receives input focus first. When the last form is closed, execution of the application is finished.

Note To open a form as a *modal window*, use the ReadModal() method or the READMODAL() function. For example, forms that serve as dialog boxes are modal, since they halt program execution until the user inputs a response.

The OPEN FORM command is equivalent to the Open() method.

Example

The following example defines four default sized forms and uses the OPEN FORM command syntax options to establish an initial focus order:

```
DEFINE FORM Opn1
DEFINE FORM Opn2 AT 4,4
DEFINE FORM Opn3 AT 8,8
DEFINE FORM Opn4 AT 12,12
OPEN FORM Opn4, Opn1, Opn3, Opn2
```

Portability

Not supported in dBASE IV or dBASE III PLUS.

See Also

Open(), READMODAL(), ReadModal()

ORDER()**Table organization**

Returns the name of the master .NDX file or the master .MDX index tag in the current or a specified work area.

Syntax

ORDER([<alias>])

<alias> A work area number (1 through 225), letter (A through J), or alias name. The work area letter or alias name must be enclosed in quotes.

Description

ORDER() returns the name of the master .NDX file or .MDX index tag in the current or specified work area. ORDER() returns an empty string ("") if no master index exists in the specified work area.

You can use ORDER() to determine the name of the master index and save the name of the index to a memory variable, and then use the SET ORDER command to later restore the master index.

Example

The following example uses ORDER() to show the name of master index for the current table:

```
USE Company EXCLUSIVE
INDEX ON Company TAG Company
INDEX ON City TAG City
SET ORDER TO Company
? "The master index file is " + ORDER() + " for the " + DBF() + " table"
SET ORDER TO      && no index
? "The master index file is " + ORDER() + " for the " + DBF() + " table"
SET ORDER TO City
? "The master index file is " + ORDER() + " for the " + DBF() + " table"
```

Portability

Not supported in dBASE III PLUS.

See Also

ALIAS(), KEY(), MDX(), NDX(), SELECT(), SET INDEX, SET ORDER, TAG(), USE

Returns the name and version number of the current operating system or the current version of Windows.

Syntax

OS([<expN>])

<expN> Any number. OS() without <expN> returns the name and version number of the operating system. OS(expN) returns the version of Windows currently running.

Description

Use OS() to determine the operating system or version of Windows in which your programs are running. To determine which version of dBASE is running, use VERSION().

OS() lets you make your applications more portable from system to system. For example, if you plan to copy an application to more than one system, and if you suspect that some of these systems have DOS versions that are too old to accommodate certain sections of your code, make the execution of these sections contingent on the value returned by OS().

Example

This example uses OS() to test whether the operating system in use is Windows version 3.10 or later. It finds the number of characters to the right of 'version', separates them out and converts them to a number. This number is compared to 3.10:

```
Os=OS(1) && Will return "Windows version ??.??."
L=Len(Os)
* Number of Chars
Os=UPPER(Os)
* convert to upper case
Pos=AT("VERSION",Os)
* locate 'version' in Os
* Pos points at the 'v' in 'version'
RightSide=L-Pos-6
* get # chars after 'version'
VersionStr=Right(Os,RightSide)
VersionNum=Val(VersionStr)
* convert to number
IF VersionNum<3.10
  ? "Your current operating system is",OS()
  ? "Your current windows system is ",OS(1)
  WAIT ;
  "This program needs Windows version 3.10 or later"
ENDIF
```

Portability

The <expN> option is not supported in dBASE IV or dBASE III PLUS.

See Also
 VERSION()

PACK

Fields and records

Removes all records marked for deletion from the current table. This command has no effect when accessing Paradox or SQL tables, since records are removed immediately when they are deleted.

Syntax
 PACK

Description

Use PACK to remove records from the current table that were previously marked for deletion by the DELETE command. You need to open a table for exclusive use before using PACK.

After you execute a PACK command, the disk space used by the deleted records is reclaimed when the table is closed. All open index files are automatically re-indexed after PACK is executed. (Use REINDEX to update closed indexes.)

Space in .DBT memo files associated with deleted records is not reduced, however, when you use the PACK command. To reclaim space in a memo file, you need to use COPY to copy the original table.

Use PACK with caution. Records that have been marked for deletion but not yet eliminated with PACK can be undeleted and restored to a table using RECALL. Records eliminated with PACK are permanently lost and can't be recovered.

SET DELETED ON provides many of the benefits of PACK without actually removing records from a table. With SET DELETED ON, most commands function as if records marked for deletion had been eliminated from a table.

To permanently remove all records of the current table in one step, use the ZAP command.

P

Example

The following example creates a form with an entry field for entering a State_Prov code and a pushbutton that calls a procedure that uses PACK to permanently delete marked records from the file TEMP.DBF:

```
* Main Program
SET SAFETY OFF
USE Clients
COPY TO Temp
USE Temp EXCLUSIVE
PUBLIC mState
mState=" "           && Three Spaces
DO Cleanup
* End Main Program
```

PAD()

```
PROCEDURE Cleanup
SET PROCEDURE TO PROGRAM(1) ADDITIVE
DEFINE FORM Pack FROM 2,2 TO 20,30
DEFINE ENTRYFIELD State OF Pack AT 7,10;
    PROPERTY Datalink "mState"
DEFINE TEXT State2 OF Pack AT 5,6;
    PROPERTY Text "Enter State_Prov Code", Width 20
DEFINE PUSHBUTTON Exit OF Pack AT 12,10;
    PROPERTY TEXT "Pack", OnClick GetRid, Width 9
OPEN FORM Pack

* Procedure for Pushbutton
PROCEDURE GetRid
CLOSE FORMS Pack
DELETE FOR State_Prov = UPPER(mState)
PACK
COUNT FOR State_Prov= UPPER(mState) TO Check
IF Check = 0
    ? "Pack was successful."
ENDIF
RETURN
```

See Also

DELETE, RECALL, SET DELETED, ZAP

PAD()

dBASE IV menus

Returns the name of the currently selected (highlighted) or most recently chosen pad in a dBASE IV menu bar. This command is supported primarily for compatibility with dBASE IV. In *Visual* dBASE, use INSPECT() to return information associated with objects in forms.

For complete syntax information on PAD(), see online Help. For information about defining forms, see the Forms chapters in the *User's Guide*.

PADPROMPT()

dBASE IV menus

Returns the prompt of the currently selected (highlighted) or most recently chosen pad in a dBASE IV menu bar. This command is supported primarily for compatibility with dBASE IV. In *Visual* dBASE, use INSPECT() to return information associated with objects in forms.

For complete syntax information on PADPROMPT(), see online Help. For information about defining forms, see the Forms chapters in the *User's Guide*.

PARAMETERS

Programs

Assigns variable names to data passed from a calling program. This command is included primarily for compatibility with prior versions of dBASE, where using PARAMETERS was the only way to assign variable names in procedures. In *Visual* dBASE, you should assign variable names by including them in parentheses after a PROCEDURE statement, because this makes them local in scope. For more information, see PROCEDURE.

PAYMENT()

Numeric data

Returns the periodic amount required to repay a debt.

Syntax

PAYMENT(<principal expN>, <interest expN>, <term expN>)

<principal expN> The original amount to be repaid over time.

<interest expN> The interest rate per period expressed as a positive decimal number. Specify the interest rate in the same time increment as the term.

<term expN> The number of payments. Specify the term in the same time increment as the interest.

Description

Use PAYMENT() to calculate the periodic amount (payment) required to repay a loan or investment of <principal expN> amount in <term expN> payments. PAYMENT() returns a float based on a fixed interest rate compounding over a fixed length of time.

If <principal expN> is positive, PAYMENT() returns a positive number.

If <principal expN> is negative, PAYMENT() returns a negative number.

Express the interest rate as a decimal. For example, if the annual interest rate is 9.5%, <interest expN> is .095 (9.5/100) for payments made annually.

Express <interest expN> and <term expN> in the same time increment. For example, if the payments are monthly, express the interest rate per month, and the number of payments in months. You would express an annual interest rate of 9.5%, for example, as .095/12, which is 9.5/100 divided by 12 months.

The formula dBASE uses to calculate PAYMENT() is as follows:

$$pmt = princ * \frac{int * (1 + int)^{term}}{(1 + int)^{term} - 1}$$

where int = rate/100

For the monthly payment required to repay a principal amount of \$16860.68 in five years, at 9% interest, the formula expressed as a dBASE expression looks like this:

P

PAYMENT()

```
? PAYMENT(16860.68,.09/12,60)      && Returns 350.00
nTemp = (1 + .09/12)^60
? 16860.68*(.09/12*nTemp)/(nTemp-1) && Returns 350.00
```

Use SET DECIMALS to set the number of decimal places PAYMENT() displays.

Example

The following example creates a form for user entered mortgage loan information and uses PAYMENT() to calculate monthly loan payments:

```
LOCAL f
f=NEW Mortgage()
f.OPEN()
CLASS Mortgage OF FORM
  this.Top=5
  this.Left=50
  this.Width=55
  this.Height=13
  this.Text="Dewey's Loans"
  DEFINE TEXT Txt1 OF THIS AT 2, 5;
    PROPERTY Text "What is the principal?";
    Width 26, ColorNormal "RB/W"
  DEFINE ENTRYFIELD Princ OF THIS AT 2,35;
    PROPERTY Value 0, ;
    Width 7
  DEFINE TEXT Txt2 OF THIS AT 4, 5;
    PROPERTY Text "What is the interest rate?";
    Width 35, ColorNormal "RB/W"
  DEFINE ENTRYFIELD Int OF THIS AT 4,35 ;
    PROPERTY Value 0, Picture "99.99";
    Width 8
  DEFINE TEXT Txt3 OF THIS AT 6,5;
    PROPERTY Text "What is the number of payments?";
    Width 50, ColorNormal "RB/W"
  DEFINE ENTRYFIELD Pay OF THIS AT 6,45 ;
    PROPERTY Value 0, Picture "999";
    Width 8
  DEFINE PUSHBUTTON Results OF THIS AT 10,18;
    PROPERTY Text "Payment";
    OnClick {myResult="Your payment will be: $" + ;
    LTRIM(STR(PAYMENT(Form.Princ.Value,;
    (Form.Int.Value/100)/12,Form.Pay.Value),13,2));
    ; Form.FV.Text = MyResult},;
    Height 2, ColorNormal "N/W", Width 15
  DEFINE TEXT FV OF THIS AT 8,8;
    PROPERTY Text "Your payment: ", Width 40,;
    ColorNormal "R+/W"
ENDCLASS
```

Portability

Not supported in dBASE III PLUS.

See Also

FV(), PV(), SET DECIMALS

PCOL()

Printing

Returns the printing column position of a printer. Column numbers begin at 0.

Syntax

PCOL()

Description

Use PCOL() to determine the horizontal printing position of a printer—that is, the column at which the printer is set to begin printing. Use PCOL() in mathematical statements to direct the printer to begin printing at a position relative to its current column position. For example, PCOL() + 5 represents a position five columns to the right of the current position, and PCOL() – 5 represents a position five columns to the left of the current position.

When you direct output to the printer, dBASE maps each character according to the *coordinate plane*, a two-dimensional grid. The coordinate plane is divided into character cells whose widths depend on the value of `_ppitch`. See the table in the description of `_ppitch`, which lists `_ppitch` values. The height of each character cell is determined by the size of the font of the parent form window. For more information about the coordinate plane, see Chapter 16 in the *Programmer's Guide*.

PCOL() returns a column number that reflects the current value of `_ppitch`, regardless of whether you're printing with proportional or monospaced fonts. If you're printing with a proportional font, you can add and subtract fractional numbers to and from the PCOL() value to move the printing position accurately. If you issue ? without the STYLE option and use only integer coordinates, dBASE uses a monospaced font and all output should appear exactly the same as in dBASE IV.

SET PRINTER must be ON or SET DEVICE TO PRINTER must be in effect for PCOL() to return a column position; otherwise, it returns 0.

Example

The following example writes "Jack & Jill" to the printer. It uses PCOL() to note the column position three times, at the beginning, after "Jack", and after "Jill":

```
SET TALK OFF
SET PRINTER ON
* now ?s are directed to printer
?          && sets printer at col 0 of next line
beginpos=pcol()  && note the current column
?? "Jack"
lastjackpos=pcol()
?? " & Jill"
lastjillpos=pcol()
SET PRINTER OFF
CLOSE PRINTER
? beginpos          && 0.00
? lastjackpos       && 4.00
? lastjillpos       && 11.00
SET TALK ON
```

P

PCOUNT()

See Also

COL(), PROW(), SET DEVICE, SET PCOL, SET PRINTER

PCOUNT()

Programs

Returns the number of parameters passed by a calling command to the called program, procedure, or user-defined function (UDF).

Syntax

PCOUNT()

Description

Because you can pass a program, procedure, or UDF more or fewer parameters than the **PARAMETERS** <parameter list> statement specifies, you can use PCOUNT() to determine exactly how many parameters have been passed.

PCOUNT() returns 0 if no parameters are passed.

Example

The following example uses PCOUNT() to determine the number of parameters passed to a procedure.

```
Param1 = 'Hello'
Param2 = 100
DO MyProg WITH Param1, Param2

PROCEDURE MyProg
PARAMETERS Par1, Pr2, Pr3, Pr4
? 'You passed ' + LTRIM(STR(PCOUNT())) + ;
" parameters to this procedure."
RETURN
```

Portability

PCOUNT() is not supported in dBASE III PLUS.

See Also

DO, FUNCTION, PARAMETERS, PROCEDURE, SET PROCEDURE

PI()

Numeric data

Returns the approximate value of pi, the ratio of a circle's circumference to its diameter.

Syntax

PI()

Description

PI() returns a float that is approximately 3.141592653589793. pi is a constant. Use it to measure the area and circumference of a circle or the volume of a cone or cylinder. For example, use it in the trigonometric functions SIN(), COS(), and TAN(), which require the angle value in radians. pi radians is equivalent to 180 degrees.

Use SET DECIMALS to set the number of decimal places PI() displays.

Example

The following example uses PI() in a program to determine geometric dimensions of a hot tub:

```
SET TALK OFF
CLEAR
width = 0
depth = 0
@ 12,12 SAY "Enter the width of your hottub " GET width PICTURE "99"
@ 13,12 SAY "What is the depth of the hottub" GET depth PICTURE "99"
READ
@ 15,12 SAY "The hottub has a circumference of " + ;
    LTRIM(STR((2 * PI() * width/2))) + " feet"
@ 16,12 SAY "The hottub has a volume of " + ;
    STR(PI() * ((width/2)^2) * depth,5,2) + " cubic feet"
@ 17,12 SAY "The hottub is " + ;
    STR(PI() * ((width/2)^2),5,2) + " feet in area"
```

Portability

Not supported in dBASE III PLUS. In dBASE IV, PI() returns a value accurate only to 14 decimal places, regardless of the value of SET DECIMALS.

See Also

COS(), DTOR(), RTOD(), SET DECIMALS, SIN(), TAN()

PLAY SOUND

Data objects

P

Plays a sound stored in a .WAV file or a binary field.

Syntax

PLAY SOUND

FILENAME

<filename> | ? | <filename skeleton> |

BINARY <binary field>

FILENAME <filename> | ? | <filename skeleton> |

BINARY <binary field> Specifies the sound file or binary field. PLAY SOUND FILENAME ? and PLAY SOUND FILENAME <filename skeleton> display a window that lets the user select a sound file. <filename> is the name of a sound file; PLAY SOUND assumes a .WAV extension unless you specify otherwise. If you specify a file without including its path, dBASE looks for the file in the current directory, then in the path you specify with

PLAY SOUND

SET PATH. PLAY SOUND BINARY <*binary field*> plays the sound stored in a binary field.

Description

Use PLAY SOUND to play sounds you record with the Windows Sound Recorder or other sound applications developed for Windows. These sounds may include music, speech, or any other sounds stored in binary fields or in .WAV files.

Note To play a sound, your system must have a sound driver or a sound adapter board.

Example

The following example uses PLAY SOUND in a codeblock of an OnClick property of pushbutton Sound to play a sound stored in a binary field:

```
LOCAL f
f = NEW PICTURES ()
f.Open()

CLASS PICTURES OF FORM
  this.EscExit = .T.
  this.View = "PICTURES.QBE"
  this.ColorNormal = "BG/B"
  this.Text = "Pictures Form"
  this.Width = 76.00
  this.Top = 0.00
  this.Left = 0.00
  this.Height = 30.00
  this.Minimize = .F.
  this.Maximize = .F.
  this.OnOpen = {;create session}

  DEFINE PUSHBUTTON SOUND OF THIS;
  PROPERTY;
  OnClick {;PLAY SOUND Binary Pictures->Sound},;
  Text "Sound",;
  Width 18.00,;
  Top 5.00,;
  Left 1.00,;
  Height 3.00,;
  FontSize 16.00,;
  FontName "Courier"
  * Additional object definitions
ENDCLASS
```

Portability

Not supported in dBASE IV or dBASE III PLUS.

See Also

SET PATH TO

POPUP()

dBASE IV menus

Returns the name of the current dBASE IV pop-up menu. This command is supported primarily for compatibility with dBASE IV. In *Visual dBASE*, use `INSPECT()` to return information associated with objects in forms.

For complete syntax information on `POPUP()`, see online Help. For information about defining forms, see the Forms chapters in the *User's Guide*.

PRINTJOB...ENDPRINTJOB

Printing

Uses the values stored in system memory variables to control a printing operation.

Syntax

```
PRINTJOB
<statements>
ENDPRINTJOB
```

<statements> Any valid dBASE statements.

Description

Use `PRINTJOB...ENDPRINTJOB` to control a printing operation with the values of the system memory variables `_pbpage`, `_pepage`, `_pcopies`, `_peject`, and `_plineno`. When dBASE begins executing `PRINTJOB`, it does the following:

- 1 Closes the current print document (if any) and begins a new one, as if you had issued `CLOSE PRINTER` before issuing `PRINTJOB`
- 2 Ejects a page if `_peject` is set to "BEFORE" or "BOTH"
- 3 Sets `_pcolno` to 0

When dBASE reaches `ENDPRINTJOB`, it does the following:

- 1 Ejects a page if `_peject` is set to "AFTER" or "BOTH"
- 2 Resets `_pcolno` to 0

Before using `PRINTJOB...ENDPRINTJOB`, set the relevant system memory variables and issue `SET PRINTER ON`. After `ENDPRINTJOB`, use `CLOSE PRINTER` to close and print the document.

Example

The following example uses `PRINTJOB` to print one line of text making three copies:

```
_pcopies=3          && 3 copies
_peject="none"      && no page eject before or after
_plineno=0          && initialized to 0
SET PRINTER ON
PRINTJOB
? "A one line print job"
?
```

P

PRINTSTATUS()

```
ENDPRINTJOB
CLOSE PRINTER      && initiate printing
* prints:
* A one line print job
*
* A one line print job
*
* A one line print job
*
```

Portability

Not supported in dBASE III PLUS.

See Also

_pbpage, _pcopies, _peject, _pepage, _plineno, EJECT, EJECT PAGE, ON PAGE, SET PRINTER

PRINTSTATUS()

Printing

Returns true (.T.) if the print device is ready to accept output.

Syntax

PRINTSTATUS([<port name expC>])

<port name expC> A character expression such as "LPT1" that identifies the printer port to check. Valid port names include all designations that the Windows Control Panel recognizes.

Description

Use PRINTSTATUS() to determine whether you've designated a printer port as an output device with SET PRINTER TO <port name expC>. In dBASE, the Windows Print Manager spools print output to and manages the printer port. Therefore, the Print Manager informs you when a printer isn't ready to receive output.

If you don't pass <port name expC> to PRINTSTATUS(), it checks the default port you specified with SET PRINTER TO. PRINTSTATUS() returns only .F. if you haven't specified a printer port with SET PRINTER TO or if the port you specify hasn't been set with SET PRINTER TO.

Note dBASE automatically executes SET PRINTER TO on startup if the WIN.INI file contains a valid printer definition. See your Windows documentation for information on WIN.INI settings.

Example

This example reads the default PRINTSTATUS and then queries LPT1, LPT2 and LPT3:

```
? PRINTSTATUS()
? PRINTSTATUS("LPT1")
? PRINTSTATUS("LPT2")
? PRINTSTATUS("LPT3")
```

Portability

Not supported in dBASE III PLUS.

See Also

CLOSE..., SET DEVICE, SET PRINTER

PRIVATE**Memory variables**

Declares memory variables that you can use in the program where they're declared and in all subroutines the program calls. PRIVATE is supported primarily for compatibility with dBASE IV. In *Visual* dBASE, declaring variables LOCAL is recommended.

Syntax

PRIVATE <memvar list> |

ALL

[LIKE <memvar skeleton 1>]

[EXCEPT <memvar skeleton 2>]

<memvar list> The list of memory variables you want to declare private, separated by commas.

ALL Makes private all memory variables declared in the subroutine.

LIKE <memvar skeleton 1> Makes private the memory variables whose names are like the memory variable skeleton you specify for <memvar skeleton 1>. Use characters of the variable names and the wildcards * and ? to create <memvar skeleton 1>.

EXCEPT <memvar skeleton 2> Makes private all memory variables except those whose names are like the memory variable skeleton you specify for <memvar skeleton 2>. Use characters of the variable names and the wildcards * and ? to create <memvar skeleton 2>. You can use LIKE and EXCEPT in the same statement, for example, PRIVATE ALL LIKE mvar* EXCEPT mvarnum*.

Description

Use PRIVATE to declare a memory variable in a subroutine private to that subroutine and its subroutines, and therefore inaccessible to higher-level routines in the call chain. To declare a variable PRIVATE, do so before initializing it to a particular value. Declaring a variable PRIVATE, however, doesn't create it. After declaring a variable private, you can create and initialize it to a value with STORE, =, or DECLARE.

A private variable is inaccessible to routines above it in the calling chain. In this way, a private variable is protected from having its value changed by routines above it in the calling chain that have variables with the same name. A variable that is private to a subroutine is in effect a different variable from one with the same name in a higher-level or unrelated routine. Once a private variable is assigned a value, DISPLAY MEMORY and LIST MEMORY indicate a variable of the same name in a higher-level subroutine as hidden.

By default, variables you initialize in programs are private. However, if you initialize a variable that has the same name as a variable declared PUBLIC in an earlier subroutine

P

or in the Command window, and don't declare the variable PRIVATE, it *not* created as a private variable. Instead, the subroutine uses and alters the value of the existing public variable.

A private variable is not protected from alteration by a lower-level subroutine that has a variable with the same name. If a lower-level subroutine stores to or alters a variable that has the same name as one to which data has been stored at a higher level, the higher-level variable is accessed, and its data overwritten.

To prevent this from happening—to have dBASE in effect consider the same-named variables to be two different variables—declare the variable PRIVATE in the lower-level subroutine (or define variables as LOCAL). If you want the lower-level subroutine to have access to the value of the variable but not be able to send back an altered value, include it in parentheses in the DO...WITH statement.

Private variables are erased from memory when the subroutine that creates them finishes executing.

For more information, see PUBLIC for a table that compares the scope and availability of public, private, local, and static variables.

Example

The following example branches from a main program to several lower-level procedures to demonstrate that a variable declared PRIVATE in Level_2 is available in all procedures below the level on which it is declared, but that it is not accessible once the program returns to the main program level:

```
* **Main.Prg**
CLOSE ALL
CLEAR ALL
CLEAR
SET TALK OFF
? ***Main.PRG***
STATIC nTotal
PUBLIC cString
nTotal = 7109.50
cString= "Hello"
ON ERROR ? "Var not available";
&& Displays message on error
? nTotal          && Returns 7109.50
? cString          && Returns "Hello"
DO Level_2         && Branch to Proc Level_2
?
? ***Back to Main.PRG***
? Deadline         && "Var not available" above Proc Level_2

PROCEDURE Level_2
?
? ***Proc Level_2*** && Orientation only
PRIVATE Deadline   && Var declared Private in Procedure Level_2
? Deadline         && "Var not available"-not yet initialized
Deadline = {12/31/99} && Initialize variable
? Deadline         && Now it has a value
DO Level_3         && Branch to Level3
```

```

?
? ***Back to Level_2***
? Deadline      && 12/31/99 still available
RETURN          && Return to line after DO Level_2 in Main

PROCEDURE Level_3
?
? ***Proc Level_3*** && Orientation only
? cString        && "Hello" is Public
? nTotal         && "Var not available"-Static in Main
? Deadline       && 12/31/99 available in Procs below Level_2
DO Level_4       && Branch to Level_4
?
? ***Back to Level_3***
? Deadline       && 12/31/99 Still available
RETURN          && Return to line after DO Level_3 in Level_2

PROCEDURE Level_4
?
? ***Proc Level_4*** && Orientation only
? nTotal         && "Var not available"-Static in Main
? cString        && "Hello" - Public variable
? Deadline       && 12/31/99 still available in Procs below Level_2
RETURN          && Return to line after DO Level_4 in Level_3

```

Portability

dBASE IV and dBASE III PLUS do not support both LIKE and EXCEPT in the same PRIVATE statement.

See Also

CLEAR MEMORY, DECLARE, LOCAL, PUBLIC, STATIC, STORE, =

PROCEDURE

Programs

P

Defines a procedure in a program file and optionally declares memory variables to represent parameters passed to the procedure.

Syntax

```

PROCEDURE <procedure name> [(<parameter list>)]
[<statements>]
[RETURN [<return exp>]]

```

<procedure name> The name of the procedure. Although dBASE imposes no limit to the length of procedure names, it recognizes only the first 32 characters. Procedure names can contain letters, numbers, and underscores.

(<parameter list>) Memory variable names to assign to data items (or *parameters*) passed to the procedure by the statement that called it. The variables in *<parameter list>* are local in scope, protecting them from modification in lower-level subroutines. For more information about the local scope, see LOCAL.

The number of variables assigned can be different from the number of parameters passed. You can use PCOUNT() to identify the number of parameters a procedure has received. You can include up to 255 variable names in *<parameter list>*.

Note Procedures written in prior versions of dBASE might contain variable names that were declared using the PARAMETERS statement. This use is supported in *Visual dBASE* for backward compatibility, but is not recommended because it scopes the variables as private instead of local. Variables scoped as private may inadvertently be overwritten if a subroutine the procedure calls uses a variable with the same name. For more information about the private scope, see PRIVATE.

<statements> Any valid program statements that you want the procedure to execute. This can include assignment statements, commands, procedure calls, dBASE function calls, and other procedure calls. You can call procedures recursively.

RETURN [*<return exp>*] Returns program control and supplies the value defined by *<return exp>* to the calling statement. If you don't include RETURN, the procedure ends when it encounters another PROCEDURE statement, a CLASS statement, a FUNCTION statement, or the end of the file. If you plan to use the value returned by the procedure, as in ? MyProc(), you must include RETURN *<return exp>*.

Description

Use PROCEDURE to define a program subroutine that carries out certain commands and optionally returns a value. With procedures, you can extend the dBASE language to perform a wide range of functions that meet particular needs in your applications. You also create more modular code, which is easier to debug and maintain.

For example, if there are several places in your application where you want to perform certain tasks, include them in a procedure, as shown in the following example. In this case, each time you want to skip a record and go to the top of the file on encountering EOF, you would issue DO NextRecOrTop.

```
PROCEDURE NextRecOrTop
  SKIP      && go to the next record if possible
  IF EOF()  && if reach end-of-file
    GO TOP  && go to first record in the table
  ENDIF
RETURN
```

Issue PROCEDURE only in a program file (.PRG or .WFM). You can't nest procedures, or begin a procedure within a processing loop defined with IF, SCAN, etc.

You call a procedure with DO or with the call operator (parentheses), as shown in the following example.

```
** Two ways to call the same procedure
x = 100
DO FirstProc WITH x
? x          && returns 200
x = 100
? Firstproc(x)  && returns 200

PROCEDURE FirstProc(mvar)
mvar = mvar * 2
RETURN mvar
```


Using parameters

Use parameters to exchange values or references between one subroutine and a subroutine it calls. For example, if you regularly need to return a date 30 days from another date, you can create a procedure to calculate and return this value using the first date as a parameter you send to the procedure. The called procedure creates a local variable name representing the value of the parameter you send, performs the calculation, and returns the value representing a date 30 days later.

This is shown in the following example. Each time you need to add 30 days to a date, call `Thirty()` with the date as a parameter. In the example, `begindate` is the name the called procedure assigns to the parameter.

```
SET DATE AMERICAN
dStartDate = {03/12/95}
? Thirty(dStartDate)  && returns the value in enddate

PROCEDURE Thirty(begindate)
*begindate is the variable name assigned to the value in dStartDate
enddate = begindate + 30
RETURN enddate
```

You can pass memory variables, properties, array names and array elements, fields, literals, or expressions as parameters to procedures.

Passing memory variables and properties

There are two ways to pass memory variables and properties, *by reference* or *by value*. This section uses the term "memory variable" to refer to both memory variables and properties.

- If you pass memory variables by reference, the called procedure has direct access to the variable. Its actions can change (overwrite) the value in that variable. Pass variables by reference if you want the called procedure to manipulate the values stored in the memory variables it receives as parameters.
- If you pass memory variables by value, the called procedure has access only to the value contained in the variable. Its actions can't change the contents of the variable itself. Pass variables by value if you want the called procedure to use the values in the variables in its calculations without changing their values in the calling subroutine. To pass a memory variable by value, enclose it in parentheses when you pass it.

P

The following example shows the differences between passing a memory variable by reference and by value.

```
X=10
? X          && X = 10
Do MyProc WITH X  && pass 10 by reference
? X          && X now = 11 (changed by MyProc)
DO MyProc WITH (X) && pass 11 by value
? X          && X still = 11 (unchanged by MyProc)
PROCEDURE MyProc(N) && N represents the value in X
? N          && returns 10 first time called, 11 second time called
N=N+1
```

```
? N      && returns 11 first time called, 12 second time called
RETURN
```

Field names take precedence over variable names. If a field and a variable have the same name and you want to pass the variable as a parameter, use `M-><memvar name>` to force use of the variable. For example, if you have a field named `CustNo` and a variable named `CustNo`, to pass the variable to the procedure `ShowCust`, use the following command:

```
DO ShowCust WITH M->CustNo
```

Passing arrays

If you use an array as a parameter, you can pass either the entire array, by passing only the array name, or individual elements of the array.

- Passing an entire array or an element of an array works the same as passing a memory variable.
- If you pass the array name without parentheses, you're making a pass by reference. The called subroutine can change values in the array.
- If you enclose the array name in parentheses when you pass it, you're making a pass by value. The called subroutine can't change values in the array.
- When you pass an array element, it is always passed by reference, even if you enclose it in parentheses; the called subroutine can change values in the array.

Passing fields

In contrast to memory variables, fields passed as parameters can't be changed by a subroutine. Fields are always passed by value, so the called procedure can't change their contents.

If you want to alter a field, store its contents to a memory variable and then execute the subroutine with that variable (for example, `DO Namechg WITH cNewname`). When control returns to the calling program, replace the field contents with the memory variable contents (for example, `REPLACE Fname WITH cNewname`).

Making procedures available

You can include a procedure in the program file that uses it, or place it in a separate program file you access with `SET PROCEDURE` or `SET LIBRARY`. If you include a procedure in the program file that uses it, you should place it at the end of the file and group it with other procedures.

A single program file can contain a total of 193 procedures. To access more than 193 procedures, use `SET PROCEDURE...ADDITIVE`. The maximum size of a procedure is limited to the maximum size of a program file.

If you plan to call a procedure with the `DO` command, as in `DO ThisProcedure`, don't give the procedure the same name as the program file that contains it. If you do, dBASE tries to run the program instead of the procedure, leading to unpredictable results.

If you plan to call a procedure with the call operator (parentheses), as in `ThisProcedure()`, don't give the procedure the same name as a built-in dBASE function. If you do, dBASE executes its built-in function instead of the procedure.

When you call a procedure, dBASE searches for it in the *search path* in *search order*. If there is more than one procedure available with the same name, dBASE runs the first one it finds. For this reason, avoid using the same name for more than one procedure. See the description of DO for an explanation of the search path and order dBASE uses.

When you create a procedure, you can use its name to determine its address (a value of type "function pointer"). Codeblocks associated with objects are anonymous procedures. For more information on working with procedures, codeblocks, and function pointers, see Chapter 4 in the *Programmer's Guide*.

Example

The following example calls procedures from within the main program file:

```
*MAIN.PRG
SET TALK OFF
CLEAR
DO A
DO B
DO C
RETURN

PROCEDURE A
@ 10,2 SAY "PROCEDURE A at the end of MAIN.PRG"
RETURN

PROCEDURE B
@ 15,2 SAY "PROCEDURE B at the end of MAIN.PRG"
RETURN

PROCEDURE C
@ 20,2 SAY "PROCEDURE C at the end of MAIN.PRG"
RETURN
```

P

The following example calls procedures from a separate procedure file:

```
*MAIN.PRG
SET TALK OFF
CLEAR
SET PROCEDURE TO PROCFILE
DO A
DO B
DO C
RETURN

*PROCFILE.PRG
PROCEDURE A
@ 10,2 SAY "PROCEDURE A in PROCFILE.PRG"
RETURN

PROCEDURE B
@ 15,2 SAY "PROCEDURE B in PROCFILE.PRG"
```

```
PROGRAM( )

RETURN

PROCEDURE C
@ 20,2 SAY "PROCEDURE C in PROCFILE.PRG"
RETURN
```

Portability

In dBASE IV, you could have a procedure and a UDF (declared with FUNCTION) with the same name available at the same time, because they were called differently. In *Visual* dBASE, if a procedure and a UDF of the same name are declared, the first one declared is the only one recognized.

The following table summarizes other differences in the use of PROCEDURE in dBASE III PLUS, dBASE IV, and *Visual* dBASE.

	dBASE III PLUS	dBASE IV	Visual dBASE
Maximum length of procedure name	8 characters	Unlimited, but only first 9 characters are recognized	Unlimited, but only the first 32 characters are recognized
Characters allowed in procedure name	Letters, numbers, and underscores; first character must be a letter	Letters, numbers, and underscores; first character must be either a letter or a number	Letters, numbers, and underscores
Maximum number of procedures per program file	32 (33 and above aren't recognized)	963	193; use SET PROCEDURE...ADDITIVE to access more than 193
Maximum procedure size	Unlimited	65520 bytes of compiled code	Same as maximum file size
Treatment of unnamed procedures	Not listed as one of the procedures in the procedure list	Listed as one of the procedures in the procedure list; by default, given the name of the program file	Same as dBASE IV
Line continuation character (;)	Can't be used to separate the PROCEDURE command and the procedure name	Can be used to separate the PROCEDURE command and the procedure name	Same as dBASE IV

See Also

CLASS, CLOSE..., COMPILE, DEBUG, DO, FUNCTION, LOCAL, PCOUNT(), PRIVATE, RETRY, RETURN, SET LIBRARY, SET PROCEDURE

PROGRAM()

Error handling and debugging

Returns the name of the currently executing program, procedure, or user-defined function (UDF).

Syntax

```
PROGRAM([<expN>])
```

<expN> Any number.

Description

PROGRAM() returns the name of the lowest level executing subroutine—program, procedure, or UDF. PROGRAM() returns an empty string ("") when no program or subroutine is executing.

PROGRAM(expN) returns the full path name of the program that is currently running, which may be different from the name of the lowest level executing subroutine. This is shown in the following example.

```
SET PROCEDURE TO program1
** Inside PROGRAM1.PRG is PROCEDURE procedure1
** If procedure1 is running, note the following:
? PROGRAM() returns PROCEDURE1
? PROGRAM(expN) returns C:\VISUALDB\PROGRAM1.PRG.
```

You can issue PROGRAM() in the Command window if a program is suspended with SUSPEND. For example, if Program A calls Procedure B, and Procedure B is suspended, issuing PROGRAM() in the Command window returns the name of Procedure B; issuing PROGRAM(expN) in the Command window returns the full path name of the file containing Procedure B.

You can also use PROGRAM() with ON ERROR and LINENO() to identify the subroutine that was executing and the exact program line number at which the error occurred.

PROGRAM() returns the name of the subroutine in uppercase letters. PROGRAM() doesn't include a file-name extension even if the subroutine is a separate file, while PROGRAM(expN) always includes a file-name extension.

Example

See the example of ON ERROR for an example of PROGRAM().

Portability

Not supported in dBASE III PLUS.

P

See Also

DEBUG, LINENO(), ON ERROR, PROCEDURE, RESUME, SET PROCEDURE, SUSPEND

PROMPT()

dBASE IV menus

Returns the prompt of the currently selected (highlighted) or most recently chosen menu item. This command is supported primarily for compatibility with dBASE IV. In Visual dBASE, use INSPECT() to return information associated with objects in forms.

For complete syntax information on PROMPT(), see online Help. For information about defining forms, see the Forms chapters in the *User's Guide*.

PROPER()

Converts a character string to proper-noun format and returns the resulting string.

Syntax

PROPER(<expC> | <memo field>)

<expC> | <memo field> The character string or memo field to convert to proper-noun format.

Description

PROPER() returns a string where the first character of each word in a character expression or memo field is capitalized and the remaining letters are lowercase. PROPER() changes the first character of a word only if it is a lowercase alphabetic character. PROPER() returns a maximum of 32766 characters, the maximum length of a string.

The current language driver defines the character values that are lowercase and uppercase alphabetic. In a U.S. language driver, a lowercase alphabetic character is from a to z, and an uppercase alphabetic character is from A to Z. See Appendix C in the *Programmer's Guide* for more information about language drivers.

Example

The following example uses PROPER() to create consistent capitalization of text strings:

```
? PROPER("e. b. white")    && Returns "E. B. White"
? PROPER("e.b. white")    && Returns "E.b. White"
? PROPER("4-WINDS MUSIC") && Returns "4-winds Music"
? PROPER("")              && Returns ""
```

When character field data is entered in all uppercase, you might want to permanently convert the field contents to upper- and lowercase. The following example replaces all uppercase company names in the Company table with the first letter of each word in uppercase and the remainder of each word in lowercase:

```
USE COMPANY
REPLACE ALL Company with PROPER(Company)
```

PROPER() also works with memo fields. The next example displays the contents of all memo fields in the Contact table and uses PROPER() to convert all first letters of strings to uppercase:

```
USE Contact
SCAN
? PROPER(Notes)
?
ENDSCAN
RETURN
```

Portability

Not supported in dBASE IV or dBASE III PLUS.

See Also

LDRIVER(), LOWER(), SET LDCHECK, UPPER()

PROTECT

Security

Creates and maintains security on a dBASE system.

Syntax

PROTECT

Description

This command is issued within dBASE by the database administrator, who is responsible for data security. PROTECT works in a single user or multiuser environment.

PROTECT is optional. If you use it, however, the security system always controls dBASE table access.

This command displays a multi-page dialog. The first time you use protect, the system prompts you to enter and confirm an administrator password.

Warning Remembering the administrator password is essential. You can access the security system only if you can supply the password. Once established, the security system can be changed only if you enter the administrator password when you call PROTECT. Keep a hard copy of the database administrator password in a secured area. There is no way to retrieve a password from the system.

PROTECT includes three distinct types of database protection:

- Log-in security, which prevents access to dBASE, or all protected tables (at the discretion of the database administrator), by unauthorized personnel.
- File and field access security, which allows you to define what dBASE tables, and fields within tables, each user can access.
- Data encryption, which encrypts dBASE tables so that unauthorized users cannot read them

The following table summarizes the database security types, how to implement each security type, and the results of security implementation.

Security Type	You Define:	You Get:
Log-in	User name and password	Control over access to dBASE or all protected tables
File and Field Access	Access levels	Control over access to dBASE tables, fields in tables, 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 the previous table.

Log-in security is the first security level. Once a security system is in place, you can set up log-in security in one of two ways:

- Users cannot access dBASE until they pass log-in security.
- Users can access dBASE, but cannot access any protected table until they pass log-in security.

Access control is the next security level. Access control determines what a user can do both with the table and the data in the table, 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 has the lowest access privileges. You establish an access level for each user in the user's profile, and additional access levels for table and field privileges in the *table privilege scheme*.

You establish privileges for a table by assigning access levels, in any combination, for read, update, extend, and delete operations.

Data encryption scrambles the table so that unauthorized users cannot read the data.

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 attempts to start dBASE (if dBASE is configured to require a log-in to start the program), or attempts to access a protected table (if dBASE is configured to require a log-in when a protected table is accessed), dBASE looks for a DBSYSTEM.DB file. You can specify a location for this file in the [CommandSettings] section of DBASEWIN.INI:

```
DBSYSTEM=C:\VISUALDB\BIN
```

If there is no DBSYSTEM entry in DBASEWIN.INI, dBASE looks for the file in the same directory in which DBASEWIN.EXE is located. If it finds the file, it initiates the log-in process. If it does not find the file, there is no log-in process.

DBSYSTEM.DB is maintained as an encrypted file. Keep a record of the information contained in DBSYSTEM.DB, as well as a current backup copy of the file. If the DBSYSTEM.DB file is deleted or damaged and no backup is available, the database administrator will need to reinitialize PROTECT using the same administrator password and group names as before, or the data will be unrecoverable.

See the "Restricting access to confidential tables" chapter in the *User's Guide* for more information about PROTECT.

See Also

ACCESS(), LOGOUT, SET ENCRYPTION, USER()

PROW()

Printing

Returns the printing row position of a printer. Row numbers begin at 0.

Syntax

PROW()

Description

Use PROW() to determine the vertical printing position of a printer—that is, the row at which the printer is set to begin printing. Use PROW() in mathematical statements to direct the printer to begin printing at a position relative to its current row position. For example, PROW() + 5 represents a position five rows below the current position and PROW() – 5 represents a position five rows above the current position.

When you direct output to the printer, dBASE maps each character according to the *coordinate plane*, a two-dimensional grid. The coordinate plane is divided into character cells whose widths depend on the value of `_ppitch`. See the table in the description of `_ppitch`, which lists `_ppitch` values. The height of each character cell is determined by the size of the font of the parent form window. For more information about the coordinate plane, see Chapter 16 in the *Programmer's Guide*.

If you're printing with a proportional font, you can add and subtract fractional numbers to and from the PROW() value to move the printing position accurately. If you issue ? without the STYLE option and use only integer coordinates, dBASE uses a monospaced font, and all output appears exactly the same as in dBASE IV.

SET PRINTER must be ON or SET DEVICE TO PRINTER must be in effect for PROW() to return a row position; otherwise, it returns 0.

Example

This example reads the current row from PROW() and writes "hello" two rows down and some results on the line below it:

```
SET TALK OFF
SET DEVICE TO PRINTER
EJECT    && prow() is now zero
Beginrow=prow()
@ prow()+2,0 SAY "Hello"
* prow() is now beginrow+2
*   because of the @SAY command
Hellorow=prow()
@ prow()+1,0 SAY "Begin row="          && move to next row
@ prow(),pcol() SAY Beginrow          && 0.00
@ prow(),pcol() SAY " Hello row="
@ prow(),pcol() SAY Hellorow          && 2.00
@ prow(),pcol() SAY " Prow()="
@ prow(),pcol() SAY prow()           && 3.00
SET DEVICE TO SCREEN                  && reset device
CLOSE PRINTER                        && Initiates the printout
```

P

See Also

PCOL(), ROW(), SET PROW

Declares global memory variables or arrays that you can use and change in any dBASE program or subroutine, or in the Command window.

Syntax

```
PUBLIC <memvar list> |
ARRAY <array name 1>["<expN list 1>"]
[, <array name 2>["<expN list 2>"]...]
```

Brackets ([]) in double quotation marks are required syntax components.

<memvar list> The memory variables to make public.

ARRAY <array name 1>[<expN list 1>], <array name 2>[<expN list 2>] ... The array variable(s)—<array name 1>, <array name 2>, and so on—and array element(s) of each array—<expN list 1>, <expN list 2>, and so on—to make public.

Description

Use PUBLIC to declare a memory variable, including array variables, in a subroutine accessible to higher-level and lower-level routines in a program and to the Command window.

When control passes from a *subroutine*—a program, procedure, or user-defined function (UDF)—to the higher-level calling routine—a program, procedure, UDF, or the Command window—dBASE normally clears from memory any variables the subroutine initialized. Declaring a variable PUBLIC prevents that variable from being cleared when control passes to the higher-level routine.

To declare a variable PUBLIC, do so before initializing it to a particular value. Declaring a variable PUBLIC creates it and initializes it to .F.

By default, variables you initialize in the Command window are public, and those you initialize in programs are private. The following table compares the characteristics of variables declared PUBLIC, PRIVATE, LOCAL and STATIC in a subroutine called CreateVar.

	PUBLIC	PRIVATE	LOCAL	STATIC
Created when it is declared and initialized to a value of .F.	Y	N	N	Y
Can be used and changed in CreateVar	Y	Y	Y	Y
Can be used and changed in lower-level subroutines called by CreateVar	Y	Y	N	N
Can be used and changed in higher-level subroutines that call CreateVar	Y	N	N	N
Automatically released when CreateVar ends	N	Y	Y	N

By default, when you have a program suspended, dBASE assigns private status to memory variables, including arrays, that you initialize in other programs and in the Command window. When you don't have a program suspended, dBASE assigns public status to variables that you initialize in the Command window.

Declaring an array variable with PUBLIC ARRAY achieves the same result as issuing PUBLIC <array name> followed by DECLARE <array>. For example, the following first line of code achieves the same result as the following second two lines of code:

```
PUBLIC ARRAY Row[5]

PUBLIC Row
DECLARE row[5]
```

Example

The following example uses PUBLIC to make the variables Val,Val1,Val2 accessible throughout all called programs and procedures:

```
RELEASE Val,Val1,Val2
PUBLIC Val,Val1,Val2
? Val,Val1,Val2  &&  .F.  .F.  .F.
Val=10
? Val,Val1,Val2  &&  10  .F.  .F.
DO Proc1
? Val,Val1,Val2  &&  1  11  .F.
DO Proc2
? Val,Val1,Val2  &&  2  11  22

PROCEDURE Proc1
Val = 1
Val1 = 11
RETURN

PROC Proc2
Val = 2
Val2 = 22
RETURN
```

If the PUBLIC statement is omitted then Val1 and Val2 will not be available in the main program because they are first used in Proc1 and Proc2. Val would be available to Proc1 and Proc2 because it is first used in the main program.

Portability

The array name argument is not supported in dBASE III PLUS.

See Also

CLEAR MEMORY, DECLARE, LOCAL, PARAMETERS, PRIVATE, RELEASE, RESTORE, SAVE, STATIC, STORE

P

PUTFILE()

Disk and file utilities

Displays a dialog box within which the user can create a new file name. Returns the file name the user enters, or returns an empty string if the user chooses the Cancel button or presses *Esc*.

Syntax

```
PUTFILE([(<title expC>
[, <filename expC>
[, <extension expC>
[, <filetype expL>
[, <change filetype expL>]]]])
```

<title expC> A title that is displayed at the top of the dialog box.

<filename expC> The default file name that is displayed in the dialog box's entryfield. Without *<filename expC>*, PUTFILE() displays an empty entryfield.

<extension expC> A default extension for the file name that PUTFILE() returns.

<filetype expL> A logical value that determines whether the dialog box opens with a list of all file types (.T.) or with a list of tables in a database (.F.). The default is .T. If you want to specify a value for *<filetype expL>*, you must also specify a value or empty string (") for *<filename skeleton>*, *<title expC>*, and *<extension expC>*.

<change filetype expL> A logical value that determines whether the user can switch between database tables and all file types while in the dialog box (.T.) or cannot switch between file types (.F.). The default is .F. If you want to specify a value for *<change filetype expL>*, you must also specify a value or empty string (") for *<filename skeleton>*, *<title expC>*, and *<extension expC>*, and you must specify a value for *<filetype expL>*.

Description

Use PUTFILE() to generate a new file name. Once the file name is generated, you can use it in other commands and function calls. For example, you can use PUTFILE() to generate a table file name, then give another file that file name with RENAME.

By default, the dialog box opened with PUTFILE() displays file names from the directory that was last accessed through the dialog box. Each time you use the dialog box to access a different directory, that directory serves as the default the next time the dialog box is opened.

Example

The following examples use PUTFILE():

```
F1=PUTFILE()  && Simply opens the dialog box
F2=PUTFILE("Select a file name for this report")
* Adds a title to the dialog box
F3=PUTFILE("Enter report name","Report.txt")
* User OKs the default name or enters another name
F4=PUTFILE("Enter a table name","Temp",".dbf")
* Default name: temp and default extension: .dbf
? "F1",F1
? "F2",F2
? "F3",F3
? "F4",F4
```

Portability

Not supported in dBASE IV or dBASE III PLUS.

See Also

FILE(), GETFILE(), RENAME

PV()**Numeric data**

Returns a float that is the present value of an investment.

Syntax

PV(<payment expN>, <interest expN>, <term expN>)

<payment expN> The amount of the periodic payment. Specify the payment in the same time increment as the interest and term. The payment can be negative or positive.

<interest expN> The interest rate per period expressed as a positive decimal number. Specify the interest rate in the same time increment as the payment and term.

<term expN> The number of payments. Specify the term in the same time increment as the payment and interest.

Description

PV() is a financial function that calculates the original principal balance (present value) of an investment. PV() returns a float that is the amount to be repaid with equal periodic payments at a fixed interest rate compounding over a fixed length of time. For example, use PV() if you want to know how much you need to invest now to receive regular payments for a specified length of time.

Express the interest rate as a decimal. For example, if the annual interest rate is 9.5%, <interest expN> is .095 (9.5 / 100) for payments made annually.

Express <payment expN>, <interest expN>, and <term expN> in the same time increment. For example, if the payment is monthly, express the interest rate per month, and the number of payments in months. Express an annual interest rate of 9.5%, for example, as .095/12, which is 9.5/100 divided by 12 months.

The formula dBASE uses to calculate PV() is as follows:

$$pv = pmt * \frac{(1 + int)^{term} - 1}{int * (1 + int)^{term}}$$

where int = rate 100

For the present value of an investment earning 9% interest, to be paid at \$350 monthly for five years, the formula expressed as a dBASE expression looks like this:

```
? PV(350, .09/12, 60)           && Returns 16860.68
nTemp = (1 + .09/12)^60
? 350*(nTemp-1)/(.09/12*nTemp)  && Returns 16860.68
```

In other words, you have to invest \$16,860.68 now into an account paying an interest rate of 9% annually to receive \$350/month for the next five years.

Use SET DECIMALS to set the number of decimal places PV() displays.

Example

The following example uses PV() to calculate the size of a mortgage loan that the user could afford, given the monthly payment the user desires, current interest rates and the number of payments (360 for 30-year mortgage):

```

LOCAL f
f=NEW PV()
f.OPEN()
CLASS PV OF FORM
  this.Width=50
  this.Height=15
  this.Text= "What can I afford?"
  this.ColorNormal="BG+/BG"
  DEFINE TEXT Txt1 OF THIS AT 3,5;
    PROPERTY Text "Desired monthly payment?", Width 26
  DEFINE ENTRYFIELD Amnt OF THIS AT 3,38 ;
    PROPERTY Value 0, Picture "9999", Width 5
  DEFINE TEXT Txt2 OF THIS AT 5,5;
    PROPERTY Text "Current interest rate?", Width 26
  DEFINE ENTRYFIELD Int OF THIS AT 5,38 ;
    PROPERTY Value 0, Picture "99.99", Width 5
  DEFINE TEXT Txt3 OF THIS AT 7,5;
    PROPERTY Text "How many monthly payments?", Width 30
  DEFINE ENTRYFIELD Pymts OF THIS AT 7,38 ;
    PROPERTY Value 0, Picture "999", Width 4
  DEFINE PUSHBUTTON Results OF THIS AT 11,18;
    PROPERTY Text "Loan Amount", Width 15;;
    OnClick {myResult="You can afford to borrow $"+;
      LTRIM(STR(PV(Form.Amnt.Value;;
        (Form.Int.Value/100)/12,Form.Pymts.Value),13,2));
      ; Form.PV.Text=myResult}, Height 2, ColorNormal "N/W"
  DEFINE TEXT PV OF THIS AT 9,8;
    PROPERTY Text "How much to borrow? ", Width 30
ENDCLASS

```

Portability

Not supported in dBASE III PLUS.

See Also

FV(), PAYMENT(), SET DECIMALS

QUIT**Programs**

Closes all open files, clears all memory variables, exits dBASE, and returns control to the operating system.

Syntax

QUIT [WITH <expN>]

WITH <expN> Passes a return code, <expN>, to the operating system when you exit dBASE.

Description

Use QUIT to end your dBASE work. If you include QUIT in a program file, dBASE halts the program's execution and exits dBASE. To end a program's execution without leaving dBASE, use CANCEL or RETURN.

Use QUIT WITH *<expN>* to pass a return code to Windows or to another application.

Example

The following example shows how to create pushbuttons on a form that present the user with the option to terminate dBASE by issuing the QUIT command. In this example, clicking on the Exit pushbutton issues a code block containing the QUIT command:

```
DEFINE FORM QuitTest FROM 2,2 TO 15,40;
  PROPERTY Text "Testing Quit"
DEFINE PUSHBUTTON PB1 OF QuitTest AT 8,11;
  PROPERTY Text "Exit dBASE", Width 15,;
  OnClick {;QUIT}
OPEN FORM QuitTest
```

Portability

The WITH *<expN>* option is not supported in dBASE III PLUS.

See Also

CANCEL, CLEAR, CLOSE..., RELEASE, RETURN, RUN, RUN()

RANDOM()

Numeric data

Returns a decimal value between 0 and 1.

Syntax

RANDOM([*<expN>*])

<expN> The numeric or float number with which to *seed* RANDOM().

Description

Use RANDOM() to generate a series of random numbers. If you specify a positive *<expN>* value, RANDOM() uses *<expN>* as a seed value. A seed value is the value that a function operates on to return a result. Thus, RANDOM() always returns the same number when you specify the same positive *<expN>*. For example, RANDOM(199) always returns 0.11, RANDOM(399) always returns 0.23, and RANDOM(599) always returns 0.35.

If you don't specify *<expN>*, or use zero, RANDOM() uses as a seed value the number that the previous RANDOM() returned or, if there is no previous RANDOM(), a fixed internal seed value of 0.

Use RANDOM() with a negative *<expN>* value when you want RANDOM() to return a more truly random series of numbers. If you specify a negative *<expN>* value, RANDOM() uses a seed value based on the number of seconds on your computer

R

RANDOM()

system clock. As a result, using RANDOM() with negative <expN> values makes it likely that RANDOM() will return different numbers each time your program executes.

You can use RANDOM() to generate a seemingly random series of numbers while maintaining control over what those numbers are. For example, using RANDOM(5), and then RANDOM() three times without the <expN> option always returns 0.03, 0.49, 0.56, 0.68. In the first use of RANDOM(), 5 is the seed value; in the second use of RANDOM(), 0.03 is the seed value, since the previous RANDOM() returned 0.03.

Use SET DECIMALS to set the number of decimal places RANDOM() displays.

Example

The following program generates a random list of 20 flights from the Flights table for making random inspections:

```
USE FLIGHTS
SET TALK OFF
CLEAR
Count=1
DO WHILE COUNT <=20
    Flt=INT(RANDOM()*RECCOUNT()+1)
    GOTO Flt
    ? Flight_No AT 10, Origin, Dest
    Count=Count+1
ENDDO
X=INKEY(5)      && Delays display 5 seconds
CLOSE ALL
```

The following example uses RANDOM() to generate a random number to be used as part of a unique table name:

```
CLEAR
Tmp_file = "TMP" + LTRIM(STR(RANDOM()*10000,5,0))+".DBF"
USE Clients
COPY STRUCTURE TO &Tmp_file
USE &tmp_file IN 2
SELECT 2
? "The current table is: " + Tmp_file
X=INKEY(5)      && Delays display 5 seconds
CLOSE ALL
ERASE &tmp_file
```

The next example uses RANDOM() to generate three sets of six randomly picked numbers from a possible range of 1–51:

```
CLEAR
SET TALK OFF
Play=1
Pick=1
DO WHILE Play<=3
    DO WHILE Pick<=6
        lotto = RANDOM()*51
        ? "Pick"+LTRIM(STR(Pick))+ " " AT 10, ;
        LTRIM(STR(lotto,5,0))
        Pick=Pick+1
    ENDDO
    Play=Play+1
ENDDO
```



```

?
Pick=1
Play=Play+1
ENDDO
SET TALK ON
RETURN

```

Portability

Not supported in dBASE III PLUS. In dBASE IV, the function is RAND(); RANDOM() isn't recognized.

In dBASE IV, you can reset to the default seed value by issuing RAND(100001). *Visual* dBASE does not support resetting to the default seed value. Also in dBASE IV, RAND(0) always returns the same value. In *Visual* dBASE, RANDOM(0) is identical to RANDOM(), and uses as a seed value the number that the previous RANDOM() returned.

See Also

GENERATE, SET DECIMALS

RAT()

String data

Returns a number that represents the starting position of a string within another string or memo field. RAT() searches back from the right end of the target string or memo field, and returns a value counting from the beginning of the target.

Syntax

RAT(<search expC>, <target expC> | <target memo field>
[, <nth occurrence expN>])

<search expC> The string to search for in <target expC> or <target memo field>.

<target expC> | <target memo field> The string or memo field, or *target*, in which to search for <search expC>.

<nth occurrence expN> Which occurrence of <search expC> to find. If you don't specify <nth occurrence expN>, dBASE searches for the first occurrence from the end. You can search for other occurrences by specifying the number (based on starting from the end), which must be greater than zero.

R

Description

Use RAT() to search for the first or <nth occurrence expN> occurrence of <search expC> in a target string or memo field, searching right to left, end to beginning. The search is case-sensitive. Use UPPER() or LOWER() to make the search case-insensitive.

Even though the search starts from the end of the target string or memo field, the result RAT() returns represents the numeric position of <search expC> counting from the *beginning* of the target. This is shown in the following example.

```
? RAT("abc","abcdefabc")  && returns 7
**
**      |
** The bar above shows the first occurrence of 'abc'
** searching from the end of the target
** That character is in position 7 counting from the
** beginning of the target
```

If *<search expC>* occurs only once in the target, RAT() and AT() return the same value. For example, RAT("abc","abcdef") and AT("abc","abcdef") both return 1.

RAT() returns 0 when:

- The search string isn't found
- The search string is an empty string
- The search string is longer than the target string
- The *nth* occurrence you specify with *<nth occurrence expN>* doesn't exist

To find the starting position of *<search expC>*, searching from *left to right*, beginning to end, use AT(). To learn if one string exists within another, use the substring operator (\$). See Chapter 1 for information about operators.

Example

The following example uses RAT() to locate the starting point of a passed string, starting at the right end of a second string:

```
? RAT("B","ABC")                && Returns 2
? RAT("ss","Mississippi")        && Returns 6
? RAT("ss","Mississippi",2)      && Returns 3
? RAT("Z","ABC")                 && Returns 0
? RAT("a","ABC")                 && Returns 0
? RAT("ABC","AB")                && Returns 0
? RAT("", "ABC")                 && Returns 0
? RAT("a","abc",2)               && Returns 0
```

The next example uses RAT() when displaying the contents of a field:

```
USE Clients
LIST FIELDS Company, Contact ;
    FOR RAT("COMPUTER",UPPER(Company)) > 0
CLOSE DATABASES
```

Portability

Not supported in dBASE III PLUS. dBASE IV limits a memo field search to approximately 64K of data.

See Also

AT(), LOWER(), STUFF(), SUBSTR(), UPPER()

Activates all @...GET fields and memory variables in the results pane of the Command window or the current dBASE IV window. This command is supported primarily for

compatibility with dBASE IV. In *Visual* dBASE, use DEFINE, OPEN FORM, and READMODAL() to create and activate forms.

For complete syntax information on READ, see online Help. For information about defining forms, see the Forms chapters in the *User's Guide*.

READKEY()

Keyboard and mouse events

Returns an integer that identifies the key or key combination that was pressed to terminate execution of a full-screen editing command. This command is supported primarily for compatibility with dBASE IV. In *Visual* dBASE, use properties such as OnClick to initiate actions based on how a user exits a form.

For complete syntax information on READKEY(), see online Help. For information about working with forms, see the Forms chapters in the *User's Guide*.

READMODAL()

Forms

Opens a form as a modal window and returns the name of the object that has input focus when the user submits the form.

Syntax

READMODAL(<object reference> [, <expl>])

<object reference> References the object that receives initial input focus when the form is opened.

<expl> A logical condition that determines if pressing *Esc* closes the form. The default is true (.T.).

Description

Use READMODAL() to open a form as a modal window.

A form that you open as a modal window has the following characteristics:

- While the form is open, focus can't be transferred to other forms.
- Execution of the routine that opened the form stops until the form is closed. When the form is closed, control transfers to the command line after the one that opened the form.

Many applications use modal forms as dialog boxes, which typically require users to take an action before the dialog box can be closed.

By default, READMODAL() returns the name of the object that has focus when the user submits the form; however, you can specify your own return value for READMODAL() using the WITH option of CLOSE FORMS.

READMODAL() is similar to the ReadModal() method, which also opens a form as a modal window.

R

To open a form as a *modeless window*, use the `Open()` method or the OPEN FORM command.

Example

The following example uses READMODAL() to display and enable a previously defined form:

```
SET CUAENTER OFF
DEFINE FORM Mortgage FROM 2,2 TO 15,60;
  PROPERTY Text "Dewey's Loans",MDI .F.,;
  OnClose {;SET CUAENTER ON}
DEFINE TEXT Txt1 OF Mortgage AT 2,5;
  PROPERTY Text "What is the principal?";,;
  Width 26, ColorNormal "RB/W"
DEFINE ENTRYFIELD Princ OF Mortgage AT 2,45;
  PROPERTY Picture "999,999";,;
  Width 7, Value 0
DEFINE TEXT Txt2 OF Mortgage AT 4,5;
  PROPERTY Text "What is the interest rate?";,;
  Width 30, ColorNormal "RB/W"
DEFINE ENTRYFIELD Int OF Mortgage AT 4,45 ;
  PROPERTY Picture "99.99";,;
  Width 5, Value 0
DEFINE TEXT Txt3 OF Mortgage AT 6,5;
  PROPERTY Text "What is the number of;
payments?";,Width 39,ColorNormal "RB/W"
DEFINE ENTRYFIELD Pay OF Mortgage AT 6,45 ;
  PROPERTY Picture "999";,;
  Width 4, Value 0
DEFINE TEXT Txt4 OF Mortgage AT 8,8;
  PROPERTY Text "Your payment will be:";,;
  Width 30, ColorNormal "R+/W"
DEFINE ENTRYFIELD Pymnt OF Mortgage AT 8,45;
  PROPERTY Width 10, Value 0, Picture "$9999.99";,;
  OnGotFocus {;this.Value=;
    PAYMENT(Form.Princ.Value,;
      (Form.Int.Value/100)/12,Form.Pay.Value)}
DEFINE PUSHBUTTON Exit OF Mortgage AT 11,18;
  PROPERTY Text "Exit", OnClick {;Form.Close()}
READMODAL(Mortgage.Princ)
```

Portability

Not supported in dBASE IV or dBASE III PLUS.

See Also

ACTIVATE WINDOW, CLEAR WINDOWS, CLOSE..., DEACTIVATE WINDOW, DEFINE WINDOW, `Open()`, OPEN FORM, ReadModal(), RELEASE WINDOWS

RECALL

Fields and records

Unmarks records that were previously marked for deletion in the current table. This command has no effect when accessing Paradox or SQL tables, since records are removed immediately when they are deleted.

Syntax

RECALL

[<scope>]

[FOR <condition 1>]

[WHILE <condition 2>]

<scope> The number of records to recall. RECORD <*n*> identifies a single record by its record number. NEXT <*n*> identifies *n* records, beginning with the current record. ALL specifies all records. REST specifies all records from the current record to the end of the file.

FOR <condition 1>

WHILE <condition 2> Determines which records are affected by RECALL. FOR restricts RECALL to records that meet <condition 1>. WHILE starts processing with the current record and continues with each subsequent record as long as <condition 2> is true.

Description

Use RECALL to undelete records that have been marked for deletion in the current table with DELETE but have not yet been removed with PACK. Executing DELETE doesn't erase records but identifies them as to be deleted. RECALL reverses this process, unmarking the records and fully restoring them to the table. RECALL with no options unmarks the current record only.

Records eliminated with PACK or ZAP are permanently removed and can't be recovered using RECALL.

When SET DELETED is ON, the <scope> and FOR options have no effect, and the WHILE option affects only the current record. RECALL with no options is not affected by SET DELETED; the current record is recalled.

In most cases, SET DELETED ON filters out all deleted records of the active table. It is possible to access a deleted record, though, if you refer to the record by record number. For example, if record number 4 is deleted and SET DELETED is ON, GO 4 positions the record pointer at the deleted record.

R

Example

See DELETE for an example of RECALL.

See Also

DELETE, PACK, SET DELETED, ZAP

RECCOUNT()

Returns the number of records in the current or a specified table.

Syntax

RECCOUNT([<alias>])

<alias> A work area number (1 through 225), letter (A through J), or alias name. The work area letter or alias name must be enclosed in quotes.

Description

RECCOUNT() retrieves a count of a table's records from the table header, which holds information about the table structure. In contrast, COUNT with no argument yields a record count by actually counting the table's records.

If no table is active, RECCOUNT() returns zero. RECCOUNT() includes all records, even if SET DELETED is ON or SET FILTER is in effect.

You can use RECSIZE() in combination with RECCOUNT() to determine the approximate size, in bytes, of a table. The DIR command displays the number of bytes DOS allocates to a table. DOS might not allocate the same number of bytes as the actual size of the file.

Example

The following example uses RECCOUNT() to return the number of records in each specified table:

```
USE Company IN SELECT()
USE Contact IN SELECT()
? "The table - Company - has " + ;
  LTRIM(STR(RECCOUNT("Company"),4,0)) + " records."
? "The table - Contact - has " + ;
  LTRIM(STR(RECCOUNT("Contact"),4,0)) + " records."
CLOSE ALL
```

The following example uses RECCOUNT() to initialize an array to the dimensions of Clients table and copies the contents of the table to the array:

```
SET TALK OFF
USE Clients
DECLARE Cnt2[RECCOUNT(),FLDCOUNT()]
COPY TO ARRAY Cnt2
Cnt = 1
DO WHILE Cnt <= RECCOUNT()
  ? Cnt2[Cnt,2]      && Displays company name
  Cnt=Cnt+1
ENDDO
CLOSE ALL
CLEAR ALL
SET TALK ON
```

See Also

DIR, DBF(), DISKSPACE(), DISPLAY STRUCTURE, RECNO(), RECSIZE()

RECNO()

Fields and records

Returns the current record number of the current or a specified table. For Paradox and SQL tables, this function returns a bookmark of the current position in a table.

Syntax

RECNO([<alias>])

<alias> A work area number (1 through 225), letter (A through J), or alias name. The work area letter or alias name must be enclosed in quotes.

Description

RECNO() returns the current record number of the table in the current or a specified work area. RECNO() considers all records, even if SET DELETED is ON or SET FILTER is in effect. If no table is open in the specified work area, RECNO() returns a value of 0.

If the record pointer moves past the last record (EOF) in the table, RECNO() returns a value that is one more than the total number of records in the table. If the record pointer moves before the first record (BOF) in the table, RECNO() returns a value of 1.

RECNO() also returns a value of 1 if there are no records in the table.

Example

The following example uses RECNO() to display the record number in a SCAN loop and after the record pointer is moved from EOF() to BOF():

```
SET TALK OFF
CLEAR
USE Company IN SELECT() EXCLUSIVE
? "Record Num" AT 4, "Company" AT 18
?
SCAN
  ? LTRIM(STR(RECNO())) AT 4, Company AT 18
ENDSCAN
IF EOF()
? "The End of File has been reached ;
at Record # " + LTRIM(STR(RECNO(),3,0))
ENDIF
GO TOP
SKIP -1
IF BOF()
? "The Beginning of File has been reached ;
at Record # " + LTRIM(STR(RECNO(),3,0))
ENDIF
CLOSE ALL
SET TALK ON
```

RECSIZE()

See Also

BOF(), EOF(), RECCOUNT()

RECSIZE()

Fields and records

Returns the number of bytes in a record of the current or specified table.

Syntax

RECSIZE([<alias>])

<alias> A work area number (1 through 225), letter (A through J), or alias name. The work area letter or alias name must be enclosed in quotes.

Description

RECSIZE() returns the number of bytes in a record of a table in the current or specified work area. If no table is open in the specified work area, RECSIZE() returns a value of zero.

LIST STRUCTURE and DISPLAY STRUCTURE also show the size of a table's records.

Example

The following example uses RECSIZE() to return the record length within the table structure in bytes for each specified table:

```
USE Company IN SELECT()
USE Contact IN SELECT()
? "The table - Company - has a record size of " + ;
  LTRIM(STR(RECSIZE("Company"),4,0)) + " bytes"
? "The table - Contact - has a record size of " + ;
  LTRIM(STR(RECSIZE("Contact"),4,0)) + " bytes"
CLOSE ALL
```

See Also

DBF(), DIR, DISPLAY STRUCTURE, LIST STRUCTURE, RECCOUNT(), RECNO()

REDEFINE

Objects

Changes an object definition in memory.

Syntax

```
REDEFINE <class name> <object name>
[OF <container object>]
[FROM <row, col> TO <row, col> > | <AT <row, col>]
[PROPERTY <stock property list>]
[CUSTOM <custom property list>]
[WITH <parameter list>]
```

<class name> The class of the object you want to redefine.

<object name> The object reference to the object you want to redefine.

OF <container object> Identifies the object that contains the object you want to redefine. (For most UI objects this container object is a form.)

FROM <row>, <col> TO <row>, <col> | AT <row>, <col> Specifies the initial location and size of the object within its parent form or window. FROM and TO specify the upper left and lower right coordinates of the object, respectively. AT specifies the position of the upper left corner.

PROPERTY <stock property list> Specifies values you assign to the built-in properties of the object.

CUSTOM <custom property list> Specifies new properties you create for the object and the values you assign to them. For information on custom properties, see Chapter 10 in the *Programmer's Guide*.

WITH <parameter list> Specifies the parameters you pass to the object. Declare these parameters with the PARAMETERS clause of the CLASS...ENDCLASS command.

Description

Use REDEFINE to change an existing object.

You control object characteristics with *properties*, which are memory variables contained in the object. For example, a form has height and width, so form objects have properties named Height and Width. The REDEFINE command lets you assign new values to these and other properties after you create the object with the DEFINE command or the NEW operator.

Each REDEFINE command has the same options as its associated DEFINE command, and they both give control over the same properties.

You can also change properties using the dot operator. For information on the dot operator and the syntax required to use it, see Chapter 10 in the *Programmer's Guide*.

Example

The following example creates four forms on the screen with a pushbutton in form Fourth that enables the user to REDEFINE forms First, Second and Third to screen positions that present a vertical form orientation:

R

REFRESH

```
SET PROCEDURE TO PROGRAM(1) ADDITIVE
PUBLIC FIRST, SECOND, THIRD, FOURTH
DEFINE FORM First FROM 0,0 TO 10,29
DEFINE FORM Second FROM 0,31 TO 10,58
DEFINE FORM Third FROM 13,0 TO 28,58
DEFINE FORM Fourth FROM 0,60 TO 28,74
  DEFINE PUSHBUTTON Remodel OF Fourth AT 20,1;
  PROPERTY;
  TEXT "Change Layout", Width 13,;
  OnClick Rotate
  DEFINE PUSHBUTTON Exit OF Fourth AT 23,1;
  PROPERTY;
  TEXT "Revert To", Width 13,;
  OnClick RevertTo
OPEN FORM First, Second, Third, Fourth
SET FOCUS TO Fourth

PROCEDURE RevertTo
CLOSE FORMS Third, Fourth
RETURN

PROCEDURE Rotate
CLOSE FORMS First, Second, Third
REDEFINE FORM First FROM 0,0 TO 28,17
REDEFINE FORM Second FROM 0,19 TO 28,37
REDEFINE FORM Third FROM 0,39 TO 28,58
OPEN FORM First, Second, Third
SET FOCUS TO FOURTH
RETURN
```

Portability

Not supported in dBASE III PLUS or dBASE IV.

See Also

CLASS...ENDCLASS, DEFINE

REFRESH

Table basics

Updates the current or specified work area data buffers to reflect the latest changes to data.

Syntax

REFRESH [*<alias>*]

<alias> A work area number (1 through 225), letter (A through J), or alias name. The work area letter or alias name must be enclosed in quotes.

Description

Use REFRESH to update specified work area data buffers so that data you display reflects the latest changes made to tables by other users on a network. This command is

most often used when accessing data from tables stored on a database server; however, you can also use it to update the data buffers for open Paradox and dBASE tables.

Example

This example opens Company.DBF and Orders.DBF. Because they are not exclusively opened, another user could change a record in Company or Orders after this user has read it from the server. REFRESH rereads the records to ensure that the latest data is displayed:

```
CLOSE ALL
SET EXCLUSIVE ON
USE S:\VISUALDB\Samples\Orders
INDEX ON Customer_N TAG Customer_N
* build the index
CLOSE ALL
SET EXCLUSIVE OFF
USE S:\VISUALDB\Samples\Company
SELECT 2
USE S:\VISUALDB\Samples\Orders
SET RELATION TO Customer_N INTO Company
* Now anyone can access company
SCAN
  REFRESH ("Company")
  REFRESH ("Orders")
  EDIT
ENDSCAN
```

Portability

Not supported in dBASE IV or dBASE III PLUS.

See Also

SET REFRESH

REINDEX

Table organization

Updates all open index files in the selected work area to reflect changes in the current table.

Syntax

REINDEX

Description

Use REINDEX to incorporate all changes made to the current table into the open index files if they were not open when the table was updated. Re-indexing updates all open .NDX and .MDX files opened with USE or SET INDEX.

If an index file is created using the UNIQUE option of the INDEX command, or while SET UNIQUE is ON, REINDEX rebuilds the index as a unique index. Similarly, if you

R

RELATION()

created a descending order index using the DESCENDING option of the INDEX command, the REINDEX command rebuilds the index in descending order.

Example

The following example uses REINDEX to reindex an open index file after a new record has been added with the index file open but inactive:

```
USE Company EXCLUSIVE
INDEX ON Ytd_sales TAG Ytd OF Company1
* this creates Company1.mdx with tag Ytd
USE COMPANY EXCLUSIVE
* now Company1.mdx will not be updated
APPEND BLANK
REPLACE Company WITH "Missing From Index"
* Company1.mdx is out of date
USE COMPANY INDEX Company1 EXCLUSIVE
SET ORDER TO TAG Ytd OF Company1
*
BROWSE TITLE "Record "+ltrim(str(reccount()))+" not in YTD"
* Ytd is now the index but it is out of date
* the last record cannot be found
REINDEX
* Ytd is now updated
BROWSE TITLE "Reindex includes last record"
```

See Also

INDEX, SET INDEX, SET ORDER, SET UNIQUE, USE

RELATION()

Table organization

Returns the key expression defined with the SET RELATION command for the current or specified work area.

Syntax

RELATION(<expN> [,<alias>])

<expN> The number of a relation that you want to return.

<alias> A work area number (1 through 225), letter (A through J), or alias name. The work area letter or alias name must be enclosed in quotes.

Description

RELATION() returns the expression linking tables defined from the current or specified work area with the SET RELATION command. If the table is linked to more than one table, you can specify the number of the relation you want to return. RELATION() returns an empty string ("") if no relation is set in the <expN> position.

Use RELATION() to save the key expressions of all SET RELATION settings for later use when restoring relations. To save the target table (the table into which you SET a RELATION), use the TARGET() function.

Example

The following example uses `RELATION()` to determine the key expression used to establish a relationship between two child tables, `Contact` and `Summary`, and the parent table, `Company`:

```

CLOSE ALL
CLEAR
SELECT 1
USE Customer
INDEX ON Customer_N TAG Customer_N
SELECT 2
USE Orders
SELECT 3
USE LineItem
INDEX ON Order_no TAG Order_no
SELECT 2
SET RELATION TO Order_no INTO LineItem
SET RELATION TO Customer_N INTO Customer ADDITIVE
IF LEN(RELATION(1)) > 0
    @ 9,0 SAY "Orders.dbf is related to " + ;
    TARGET(1)+ " key expression: "+RELATION(1)
IF LEN(RELATION(2)) > 0
    @ 11,0 SAY "Orders.dbf is also related to " + ;
    TARGET(2)+ " key expression: "+RELATION(2)
ENDIF
ENDIF

```

Portability

Not supported in dBASE IV or dBASE III PLUS.

See Also

`ALIAS()`, `CREATE QUERY`, `CREATE VIEW`, `CREATE VIEW...FROM`, `ENVIRONMENT`, `SELECT()`, `SET RELATION`, `SET VIEW`, `SET()`, `TARGET()`

RELEASE

Memory variables

Deletes specified memory variables and arrays, freeing memory space for additional variables.

R**Syntax**

```

RELEASE <memvar list> |
ALL
    [LIKE <memvar skeleton 1>]
    [EXCEPT <memvar skeleton 2>]

```

<memvar list> The specific memory variables to release from memory, separated by commas.

ALL Removes from memory all stored memory variables.

LIKE <memvar skeleton 1> Removes from memory all memory variables whose names are like the memory variable skeleton you specify for <memvar skeleton 1>. Use characters of the variable names and the wildcards * and ? to create <memvar skeleton 1>.

EXCEPT <memvar skeleton 2> Removes from memory all memory variables except those whose names are like the memory variable skeleton you specify for <memvar skeleton 2>. Use characters of the variable names and the wildcards * and ? to create <memvar skeleton 2>. You can use LIKE and EXCEPT in the same statement, for example, `RELEASE ALL LIKE mvar* EXCEPT mvarnum*`.

Description

Use RELEASE to clear memory variables, thus making additional space available for new ones. To remove large groups of variables, use the option ALL [LIKE <memvar skeleton 1>] [EXCEPT <memvar skeleton 2>].

If you issue `RELEASE ALL [LIKE <memvar skeleton 1>] [EXCEPT <memvar skeleton 2>]` in a subroutine—a program, procedure, or user-defined function (UDF)—dBASE releases only the private memory variables defined in that subroutine. It doesn't release memory variables declared in higher-level routines.

When control returns from a subroutine to its calling routine, dBASE clears from memory all variables initialized in the subroutine that weren't declared PUBLIC or STATIC. Thus, you don't have to release a subroutine's private variables explicitly with RELEASE in the calling routine.

Example

The following example initializes a series of memory variables with names that are a concatenation of a string and a natural order record number. After processing, Client_ID and Company field values are held in variables IDx and CompX respectively. DISPLAY MEMORY merely confirms the presence of these memory variables. RELEASE ALL LIKE demonstrates how groups of memory variables can be released from memory, as evidenced by subsequent DISPLAY MEMORY commands:

```
SET TALK OFF
SET SAFETY OFF
USE Clients EXCLUSIVE
INDEX ON Client_ID TAG Client_ID
DO WHILE .NOT. EOF()
    VAR1="ID"+LTRIM(STR(RECNO()))
    VAR2="Comp"+LTRIM(STR(RECNO()))
    STORE Client_ID TO &VAR1
    STORE Company TO &VAR2
    SKIP
ENDDO
CLEAR
DISPLAY MEMORY          && IDx and COMPx memvars
CLEAR
RELEASE ALL LIKE COMP*
DISPLAY MEMORY          && IDx only remaining
CLEAR
RELEASE ALL LIKE ID*
DISPLAY MEMORY          && all gone
CLEAR
```

```
SET TALK ON
SET SAFETY ON
```

Portability

dBASE IV and dBASE III PLUS do not support both LIKE and EXCEPT in the same RELEASE statement.

See Also

CLEAR, LOCAL, PRIVATE, PUBLIC, QUIT, RESTORE, RETURN, SAVE, STATIC

RELEASE AUTOMEM

Fields and records

Clears all stored automem variables from memory.

Syntax

```
RELEASE AUTOMEM
```

Description

When you create a set of automem variables for a table using STORE AUTOMEM, CLEAR AUTOMEM, or USE...AUTOMEM, *Visual* dBASE initializes one variable for each field, assigning each variable the same name and data type as the corresponding field. When you no longer need the automem variables, you can use RELEASE AUTOMEM to remove them from memory, making space available for other variables. RELEASE AUTOMEM releases all memory variables with the same name as fields in the current table, even variables that were not created with an AUTOMEM command.

Closing a table or moving to another work area doesn't automatically release a table's associated automem variables. *Visual* dBASE doesn't recognize a variable as an automem variable, even if it was created as an automem variable, if it doesn't have the same name as a field in the current table. Thus, when you close a table or select another work area, the associated automem variables remain in memory. After you close a table or select another work area, RELEASE AUTOMEM doesn't remove these variables from memory so you must remove them with either RELEASE or CLEAR ALL.

Example

The following sample uses CLEAR AUTOMEM to enable the user to edit AUTOMEM variables for a new record. However, a new record will only be added if the user confirms that the data is correct. RELEASE AUTOMEM is used with ON ESCAPE and READKEY() to release all automem variables from memory when the user discontinues data entry by pressing ESCAPE:

```
SET TALK OFF
CLEAR
USE Clients
ON ESCAPE RELEASE AUTOMEM
AddMoreData()
RETURN
```

R

RELEASE DLL

```
FUNCTION AddMoreData
@0,0 to 8, 70
@10,20 to 12,45
CLEAR AUTOMEM
DO WHILE .T.
    lConfirm = .F.
    @11,22 CLEAR TO 11,39
    @1,1 SAY 'ID' GET m->CLIENT_ID
    @1,10 SAY 'COMPANY' GET m->COMPANY
    @3,1 SAY 'Contact' GET m->CONTACT
    @4,1 SAY 'Address' GET m->ADDRESS
    @6,1 SAY 'City' GET m->CITY
    @6,23 SAY 'State/Province' GET STATE_PROV
    @6,54 SAY 'Zip' GET ZIP_P_CODE
    READ
    IF READKEY() = 12
        RELEASE AUTOMEM
        EXIT
    ELSE
        @11,22 SAY "Data Correct, Y-N?";
        GET lConfirm PICTURE 'Y'
        READ
        IF lConfirm
            APPEND AUTOMEM
            CLEAR AUTOMEM
        ENDIF
    ENDIF
ENDDO
RETURN .T.
```

Portability

Not supported in dBASE IV or dBASE III PLUS.

See Also

CLEAR AUTOMEM, RELEASE, STORE AUTOMEM, USE

RELEASE DLL

Windows programming

Deactivates DLL files.

Syntax

RELEASE DLL <DLL filename list>

Description

Use RELEASE DLL when you debug a DLL file or a dBASE application. For example, you must deactivate a DLL file and activate it again each time you change one of its routines.

A DLL file is a precompiled library of external routines written in non-dBASE languages such as C and Pascal. A DLL file can have any extension, although most have extensions of .DLL. You activate a DLL file with the LOAD DLL command.

Example

The following example demonstrates the command sequence for using RELEASE DLL as a trouble shooting tool:

```
LOAD DLL myDLL.DLL
* ... test DLL operation
RELEASE DLL myDLL.DLL
* ... change .DLL or C program
LOAD DLL myDLL.DLL
* ... test again
RELEASE DLL myDLL.DLL
```

Portability

Not supported in dBASE IV or dBASE III PLUS.

See Also

EXTERN, LOAD DLL

RELEASE MENUS

dBASE IV menus

Removes deactivated menus from memory. This command is supported primarily for compatibility with dBASE IV. In *Visual dBASE*, use RELEASE OBJECT to clear an object from a form.

For complete syntax information on RELEASE MENUS, see online Help. For information about defining forms, see the Forms chapters in the *User's Guide*.

RELEASE OBJECT

Objects

Removes object definitions from memory.

Syntax

RELEASE OBJECT <container reference>.<object reference>

<container reference>.<object reference> <container reference> is an object reference variable pointing to the object (usually a form) that contains the object. <object reference> is an object reference variable pointing to the object itself.

Description

Use RELEASE OBJECT to:

- remove an object when you no longer need it.
- conserve memory resources.

R

Releasing a form from memory also releases the objects it contains. Likewise, releasing a menu from memory also releases the menus it contains.

RELEASE OBJECT is identical to the Release() method.

Example

The following example defines three check boxes on form Check_It and uses RELEASE OBJECT to remove "Choice 3" check box from the form upon selection of Choice 1:

```
PUBLIC Check_It
SET PROCEDURE TO PROGRAM(1) ADDITIVE
DEFINE FORM Check_It FROM 0,0 TO 10,18
DEFINE CHECKBOX Ck1 OF Check_It AT 2,4;
  PROPERTY Text "Choice 1",;
  OnChange Exclude, Value .F.
DEFINE CHECKBOX Ck2 OF Check_It AT 4,4;
  PROPERTY Text "Choice 2", Value .F.
DEFINE CHECKBOX Ck3 OF Check_It AT 6,4;
  PROPERTY Text "Choice 3", Value .F.
OPEN FORM Check_It

PROCEDURE Exclude
IF TYPE("Check_It.Ck3")<>"U"
  RELEASE OBJECT Check_It.Ck3
ENDIF
RETURN
```

Portability

Not supported in dBASE IV or dBASE III PLUS.

See Also

CLEAR ALL, CLOSE..., Release()

RELEASE POPUPS

dBASE IV menus

Erases dBASE IV popup menus from the screen and releases them from memory. This command is supported primarily for compatibility with dBASE IV. In *Visual dBASE*, use RELEASE OBJECT to clear an object from a form.

For complete syntax information on RELEASE POPUPS, see online Help. For information about defining forms, see the Forms chapters in the *User's Guide*.

RELEASE SCREENS

Input/Output

Removes from memory all or specified variables created by SAVE SCREEN, and clears the Command window buffer. This command is supported primarily for compatibility with dBASE IV.

For complete syntax information on CLEAR SCREENS, see online Help.

RELEASE WINDOWS

dBASE IV Windows

Releases specified dBASE IV-style window definitions from memory. This command is supported primarily for compatibility with dBASE IV. In *Visual dBASE*, use CLOSE FORMS or RELEASE OBJECT to close or release a form.

For complete syntax information on RELEASE WINDOWS, see online Help. For information about defining forms, see the Forms chapters in the *User's Guide*.

RENAME

Disk and file utilities

Renames a file on disk.

Syntax

```
RENAME <filename 1> | ? | <filename skeleton 1>
TO <filename 2> | ? | <filename skeleton 2>
```

<filename 1> | ? | <filename skeleton 1> Identifies the original file (also known as the source file). RENAME ? and RENAME <filename skeleton> display a dialog box from which you can select a file to rename. If you specify a source file without including its path, dBASE looks for the file in the current directory, then in the path you specify with SET PATH. If you specify a source file without including its extension, dBASE assumes no extension.

TO <filename 2> | ? | <filename skeleton 2> Identifies the new name for the source file (also known as the target file). The ? and <filename skeleton> options display a dialog box in which you specify the name of a target file and its directory.

Description

RENAME is a utility command that lets you change the name of a file at the operating system level.

If the source file has a file-name extension, it must be specified in the command line. If the source file is not in the current directory or the path you specify with SET PATH, a path must also be included.

RENAME differs from its DOS counterpart in that wildcards do not let you rename more than one file at a time. To rename more than one file at a time using wildcards, use !, RUN, or DOS, and execute the DOS RENAME command.

If SET SAFETY is ON and a file exists with the same name as the target file, dBASE displays a dialog box asking if you want to overwrite the existing file. If SET SAFETY is OFF and a file exists with the same name as the target file, dBASE returns an error, and the target file is not overwritten.

If you specify a new directory for the target file, dBASE copies the source file to this directory. If you specify a new drive for the target file, dBASE returns an error message and the source file is not copied or renamed.

RENAME does not automatically rename a .DBT or .MDX file associated with a .DBF file. If, for example, you rename a table file that has memo fields and you do not rename

R

the associated .DBT file, an error message displays when you try to use the file. In such cases, it is best to create a new copy of the file with COPY TO.

Example

The following examples use RENAME:

```
CLOSE DATABASES
RENAME Temp.dbf TO Savit.dbf
* Temp.dbf cannot be open
RENAME Temp.dbt TO Savit.dbt
RENAME ?
* Opens dialog box
RENAME *.qbe
* Opens dialog box showing only query files
```

See Also

!, COPY FILE, DOS, RUN, SET DEFAULT, SET DIRECTORY, SET PATH, SET SAFETY

RENAME TABLE

Table basics

Changes the name of a specified table.

Syntax

```
RENAME TABLE <old table name> | ? | <filename skeleton 1>
TO <new table name> | ? | <filename skeleton 2>
[[TYPE] PARADOX | DBASE]
```

<old table name> | ? | <filename skeleton 1> Table you want to rename. RENAME TABLE ? and RENAME TABLE <filename skeleton> display a dialog box, from which you can select a table file. If you specify a file without including its path, *Visual* dBASE looks for the file in the current directory, then in the path you specify with SET PATH. If you specify a file without including an extension or specifying its type, *Visual* dBASE assumes the file type specified with the SET DBTYPE command.

You can also rename a table in a database (defined using the BDE Configuration Utility) by specifying the database as a prefix (enclosed in colons) to the name of the table, that is, *:database name:table name*. If the database is not already open, *Visual* dBASE displays a dialog box in which you specify the parameters, such as a login name and password, necessary to establish a connection to that database.

<new table name> | ? | <filename skeleton 2> New name of the table. If you rename a table in a database, you must specify the same database as the destination of the new table. Also, the new table name must be the same type as the old table. The ? and <filename skeleton> options display a dialog box, in which you specify the name of the target table and the directory to save it in.

[[TYPE] PARADOX | DBASE Specifies the type of table you want to rename, which can be a Paradox or dBASE table.

Description

Use the RENAME TABLE command to change the name of a table without exiting to the operating system. You cannot rename an open table, and the new table name cannot already exist in the same directory or database.

If you rename a table that has associated files (such as memo or index files), those files are also automatically renamed to match the new table name.

Example

The following example uses RENAME TABLE to change the name of a closed table:

```
dbf_file = "FLIGHTS.DBF"
IF SELECT() > 1
  area = 1
  go_on = .T.
  DO WHILE area < SELECT() .AND. go_on
    SELECT (area)
    IF dbf_file = DBF()
      go_on = .F.
    ENDIF
    area = area + 1
  ENDDO
ENDIF
IF go_on
  RENAME TABLE &dbf_file TO AllFlts.DBF
ENDIF
```

Portability

Not supported in dBASE IV or dBASE III PLUS.

See Also

CLOSE..., COPY, COPY FILE, USE

REPLACE

Fields and records

Replaces the contents of specified fields in the current table with data from specified expressions.

R**Syntax**

REPLACE

```
<field 1> WITH <exp 1> [ADDITIVE]
[, <field 2> WITH <exp 2> [ADDITIVE]...]
[<scope>]
[FOR <condition 1>]
[WHILE <condition 2>]
[REINDEX]
```

<field 1> WITH <exp 1> Designates fields to be replaced by data in specified expressions. Multiple fields of a record may be changed by including additional **<field n> WITH <exp n>** expressions, separated by commas.

ADDITIVE Adds text to the end of memo field text instead of replacing existing text. You can use ADDITIVE only when the specified field is a memo field in a dBASE table.

<scope> The number of records to replace. RECORD <*n*> identifies a single record by its record number. NEXT <*n*> identifies *n* records, beginning with the current record. ALL specifies all records. REST specifies all records from the current record to the end of the file.

FOR <condition 1>

WHILE <condition 2> Determines which records are affected by the REPLACE command. A FOR clause restricts REPLACE to records that meet <condition 1>. WHILE starts processing with the current record and continues with each subsequent record as long as <condition 2> is true.

REINDEX Specifies that all affected indexes are rebuilt once the REPLACE operation finishes.

Description

The REPLACE command overwrites a specified field with new data. The field you select can be any type, including memo fields. (To replace binary or OLE fields, use REPLACE BINARY and REPLACE OLE.) The field and the expression specified by the WITH clause must have the same data type. In numeric fields, the WITH expression can 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 in the assigned field width.

To change multiple fields, include additional <*field n*> WITH <*exp n*> expressions, one for each field. When <scope>, WHILE, or FOR options are used, data is replaced in all records within the scope and all records that match specified conditions.

Use the ADDITIVE option to add a character string to the end of existing memo field text. You can leave a blank space at the beginning of the string to provide proper spacing.

Be careful when replacing data in a table with a master index open when you are also using the <scope>, WHILE, or FOR options. Visual dBASE automatically updates all open index files after a REPLACE operation finishes. After replacing data that changes the value in the key field in the master index, the record and the record pointer immediately move to the position in the index based on the new value of a key. If replacement in the key field causes a record (and the pointer) to move down past other records that fall within the scope or meet the specified conditions, those records are not replaced. To make replacements to an indexed table's key field, first close the index and update the data, and then reopen the index using SET INDEX, and update the index using REINDEX.

Replacements in fields other than the key field of the master index don't affect the order of the index and can be made with <scope>, WHILE, or FOR without complications.

When replacing a numeric or float value, the new value length cannot exceed the field width; otherwise, dBASE returns a numeric overflow message. The field contents are replaced with an approximation to the new value in scientific notation, if it will fit; otherwise the field contents are replaced with asterisks, destroying stored data.

REPLACE alters data in fields of the current table unless you use *alias->field* to specify a field in an alias table.

If no relation has been set from the current table, you can replace only the current record in the alias table. If a relation has been set, you can use *<scope>*, FOR *<condition 1>*, or WHILE *<condition 2>* to replace multiple records in the alias table.

REPLACE with no options substitutes *<exp 1>* for the contents of *<field 1>* in the current record only.

An entered expression must be of the same data type as the contents of the field it is replacing, except in the case of a memo field, which can be replaced by a character expression.

Example

The following example uses REPLACE to change or enter data in TEMP.DBF, which is a copy of Clients table:

```
SET SAFETY OFF
USE Clients EXCLUSIVE
INDEX ON Client_ID TAG Client_ID
COPY TO Temp.DBF
USE Temp
REPLACE ALL Company with PROPER(Company)
* If entered in all uppercase, changes to upper/lower case.
REPLACE ALL State_Prov with UPPER(State_Prov)
* Makes two-letter state codes all uppercase if entered in upper/lower case.
REPLACE ALL Phone with STUFF(Phone,AT("-",Phone),1,"*")
* Substitutes an asterisk for the hyphen in the Phone field.
REPLACE ALL NOTES with "Georgia state tax applies";
    ADDITIVE FOR State_Prov="GA"
* Places the text sting in the Notes memo field of each Georgia record.
GO TOP
BROWSE
CLOSE ALL
SET SAFETY ON
```

See Also

APPEND, BLANK, BROWSE, CHANGE, EDIT, REINDEX, REPLACE AUTOMEM, REPLACE BINARY, REPLACE MEMO, REPLACE OLE, SET RELATION, UPDATE

R

REPLACE AUTOMEM

Fields and records

Transfers contents of memory variables into corresponding fields of the current record in the current table.

Syntax

REPLACE AUTOMEM

Description

Automem variables are memory variables that have the same name, data type, and length as the corresponding fields of the current table. Automem variables are used to hold data that will be stored in the fields of records. You can manipulate data stored in automem variables as memory variables rather than as field values, and you can validate the data before storing the data to a table.

Create a set of automem variables for the fields in a table with `USE...AUTOMEM`, `CLEAR AUTOMEM`, or `STORE AUTOMEM`. To add new records to a table and fill the fields with values from corresponding automem variables, use `APPEND AUTOMEM` or `INSERT AUTOMEM`. To update the fields of existing records with values from corresponding automem variables, use `REPLACE AUTOMEM`.

Use `REPLACE AUTOMEM` to update all the fields of a record without having to name the fields. By contrast, with the `REPLACE` command, you need to name every field you want updated.

Remember that an automem variable and its corresponding field have the same name. When a command allows an argument that could be either a memory variable or a field, *Visual* dBASE assumes the argument refers to a field. To distinguish the memory variable from the field, prefix the names of automem variables with `m->`.

`REPLACE AUTOMEM` updates the current record. It can't update all records within a specified scope or all records matching a condition, as the `REPLACE` command can with the options `<scope>`, `FOR <condition>`, and `WHILE <condition>`.

`REPLACE AUTOMEM` doesn't replace field data with data from a memory variable with the same name but of a different data type. If you try to make such a replacement, *Visual* dBASE displays an error message.

Example

The first portion of this example creates a copy of `Clients` table that is then given to a branch office or different department for changes:

```
USE Clients EXCLUSIVE
INDEX ON SUBSTR(Client_ID,1,1)+ ;
      SUBSTR(Client_ID,2,4) TAG Client_ID
COPY TO Branch1 FOR State_Prov = "CA"
! COPY Branch1.DBF B:                                && Copies to diskette
```

After `Branch1.DBF` has been updated or changed and you wish to have these changes reflected in the master `Clients` table, the following program uses `REPLACE AUTOMEM` to transfer current field values in `Branch1.DBF` to the `Clients` table:

```
SET TALK OFF
SET EXACT OFF
USE Clients EXCLUSIVE
INDEX ON SUBSTR(Client_ID,1,1)+ ;
      SUBSTR(Client_ID,2,4) TAG Client_ID
USE Branch1 IN 2
SELECT 2
GO TOP
DO WHILE .NOT. EOF()
  CLEAR AUTOMEM
```



```

STORE AUTOMEM
* Field values in current (Branch1) record stored to AUTOMEM variables
Look = SUBSTR(Client_ID,1,1)+;
    SUBSTR(Client_ID,2,4)
SELECT 1                                && switch to Clients table
SEEK Look                                && locate matching Client_ID
IF FOUND()
    REPLACE AUTOMEM                      && update Clients to Branch1
ENDIF
SELECT 2                                && back to Branch1.DBF
SKIP                                     && increment record pointer
ENDDO
SELECT 1                                && Check results by Browsing
GO TOP                                  && Clients table
BROWSE
CLOSE ALL
RETURN

```

Portability

Not supported in dBASE IV or dBASE III PLUS.

See Also

APPEND AUTOMEM, CLEAR AUTOMEM, INSERT AUTOMEM, REPLACE, STORE AUTOMEM, USE

REPLACE BINARY

Fields and records

Replaces the contents of a binary field with the contents of another binary file.

Syntax

```

REPLACE BINARY <binary field name>
FROM <filename> | ? | <filename skeleton>
[TYPE <binary type user number>]

```

<binary field name> The binary field of the current table that is replaced by the contents of <filename>.

FROM <filename> | ? | <filename skeleton> Specifies the file to copy to the binary field in the current record. If you specify a file without including its path, *Visual* dBASE looks for the file in the current directory, then in the path you specify with SET PATH. If you specify a file without including its extension, *Visual* dBASE assumes a .BMP extension; however, the file may also include types such as .BMP, .PCX, and .WAV files. The ? and <filename skeleton> options display a dialog box, in which you specify the file to copy.

[TYPE <binary type user number>] Specifies a number that can be used to identify the type of binary data being stored. Use the BINTYPE() function to retrieve the type number. The

R

range is from 1 to 32K – 1 for user-defined file types and 32K to 64K – 1 for predefined types (although any number may be specified within the allowable range).

Predefined binary type numbers	Description
1 to 32K – 1 (32,767)	User-defined file types
32K (32,768)	.WAV files
32K + 1 (32,769)	.BMP and .PCX files

Description

Use REPLACE BINARY to copy a binary file to the current record's binary field. You can copy one binary file to each binary field of each record in a table.

While dBASE memo fields may contain types of information other than text, binary fields are recommended for storing images, sound, or any other binary or BLOB type data.

Example

The following example uses COPY BINARY to copy the Boa bitmap from ANIMALS.DBF to a file named BOA.BMP. COPY STRUCTURE creates a new table with an identical structure but no records. APPEND BLANK adds a new, blank record and REPLACE BINARY copies the contents of BOA.BMP into the field Bmp of record number one of ANIM2.DBF:

```
CLOSE ALL
USE Animals
GOTO 2
COPY BINARY Bmp TO Boa.BMP
COPY STRUCTURE TO Anim2
USE Anim2
b=2**15+1  && 32k+1 for type .BMP
APPEND BLANK
REPLACE BINARY Bmp FROM "Boa.BMP" TYPE b
EDIT
* In edit, User can now click on Bmp and see the Boa
```

Portability

Not supported in dBASE IV or dBASE III PLUS.

See Also

APPEND MEMO, BINTYPE(), COPY BINARY, REPLACE MEMO, REPLACE MEMO...FROM, REPLACE OLE, RESTORE IMAGE

REPLACE FROM ARRAY

Fields and records

Transfers data stored in an array to the fields of the current record of a table.

Syntax

```
REPLACE FROM ARRAY <array name>
[<scope>]
[FOR <condition 1>]
[WHILE <condition 2>]
[FIELDS <field list>]
[REINDEX]
```

<array name> The name of the array that you want to transfer data from.

<scope> The number of records to replace with the data stored in the specified array. RECORD <*n*> identifies a single record by its record number. NEXT <*n*> identifies *n* records, beginning with the current record. ALL specifies all records. REST specifies all records from the current record to the end of the file.

FOR <condition 1>

WHILE <condition 2> Determines which records are affected by REPLACE FROM ARRAY. FOR restricts REPLACE FROM ARRAY to records that meet <condition 1>. WHILE starts processing with the current record and continues with each subsequent record as long as <condition 2> is true.

FIELDS <field list> Restricts data replacement to the fields specified by <field list>.

REINDEX Specifies that all non-master indexes are rebuilt once the REPLACE FROM ARRAY operation finishes.

Description

Use REPLACE FROM ARRAY to transfer values from an array into fields of the current table. The number specified by the last array subscript corresponds to the number of fields you can replace. The number specified by the next-to-last subscript of the array corresponds to the number of records you can replace.

When you use REPLACE FROM ARRAY with no options or just the FIELDS option, *Visual* dBASE replaces field values starting at the current record. With a multidimensional array, if there are more records than specified by the next-to-last array subscript, *Visual* dBASE replaces record data until it runs out of array values.

Use a one-dimensional array to replace field values in one record. For example, if you use DECLARE sample[3], a one-dimensional array, the command line REPLACE FROM ARRAY sample replaces up to three fields in one record.

Use a two-dimensional array to replace the field values in more than one record. A two-dimensional array is like a table with rows corresponding to records and columns corresponding to fields. For instance, if you use DECLARE sample[2,3], the command line REPLACE FROM ARRAY sample replaces up to three fields in two records.

R

REPLACE FROM ARRAY

The data types of the array must match those of corresponding fields in the table you are replacing. If the data type of an array element and a corresponding field don't match, *Visual dBASE* returns an error.

Example

The following example creates an array with 3 elements and copies 3 field values to the array for display and possible editing. If the user chooses to edit the array values, **REPLACE FROM ARRAY** is used to copy the altered field values back to the same record in the table.

```
DECLARE CompArray[3]
* Create an array with three elements
CLOSE DATABASES
ON ESCAPE RETURN
USE Company
SET FIELDS TO Company, Phone, CompCode
DO WHILE .NOT. EOF()
    COPY TO ARRAY CompArray FIELDS Company, Phone,;
        CompCode NEXT 1
    CLEAR
    ? "Company", "Phone" AT 30, "CompCode" AT 50
    ? "*****", "*****" AT 30, "*****" AT 50
    ? CompArray[1], CompArray[2] AT 30, CompArray[3];
        AT 50
    ?
    ?
    ACCEPT "Entry Correct? (Y/N) " TO Chc
    IF UPPER(Chc)="N"
        @ 10,5 SAY "Enter Company " GET CompArray[1]
        @ 11,5 SAY "Enter Phone   " GET CompArray[2]
        @ 12,5 SAY "Enter CompCode" GET CompArray[3]
        READ
        REPLACE FROM ARRAY CompArray
    ENDIF
    SKIP
ENDDO
CLOSE DATABASES
```

Portability

Not supported in dBASE III PLUS.

See Also

APPEND FROM ARRAY, COPY TO ARRAY, DECLARE

REPLACE MEMO

Fields and records

Replaces the text of a memo field with the contents of an array.

Syntax

REPLACE MEMO <memo field> WITH ARRAY <array name>
[ADDITIVE]

<memo field> The memo field where text from an array is stored.

WITH ARRAY <array name> The array whose contents replace the text in a memo field.

ADDITIVE Causes the new text to be appended to existing text. REPLACE MEMO without the ADDITIVE option causes *Visual* dBASE to overwrite any text currently in the memo field.

Description

Use the REPLACE MEMO command to replace the text of a memo field with the contents of an array. Each element of <array name> contains the data to add one line to the specified memo field. After you store text in the elements of the array, use REPLACE MEMO to store that data in a memo field.

REPLACE MEMO, when used with STORE MEMO, lets you include the text of a memo field on a data-entry screen displaying the text from automem variables and then stores the changes back into the memo field when you've finished editing.

To add text to existing text or to an empty memo field, store empty character values to an array instead of using STORE MEMO. Add text to the array and then use REPLACE MEMO, including ADDITIVE if you don't want to overwrite existing text.

Example

The following example declares an array with an element for each field in Clients table, copies the contents of record number 4 to the array, and then uses REPLACE MEMO to place the contents of array Move into the memo field of record number 10:

```
SET TALK OFF
USE Clients
DECLARE Move[FLDCOUNT()]
GOTO 4
COPY TO ARRAY Move
GOTO 10
REPLACE Notes WITH CHR(13) + ;
    "Cross Reference: " && Overwrites prior entries
REPLACE MEMO Notes WITH ARRAY Move ADDITIVE
? Notes           && Displays Notes contents
```

R

Portability

Not supported in dBASE IV or dBASE III PLUS.

See Also

APPEND MEMO, DECLARE, REPLACE, REPLACE FROM ARRAY, REPLACE MEMO...FROM, STORE MEMO

REPLACE MEMO...FROM**Fields and records**

Inserts a text file in a memo field.

Syntax

```
REPLACE MEMO <memo field> FROM <filename> | ? | <filename skeleton>
[ADDITIVE]
```

<memo field> The memo field where a text file is inserted.

FROM <file name> | ? | <filename skeleton> The file that identifies a text file. The ? and <filename skeleton> options display a dialog box in which you can specify the file whose content you want to copy to the memo field.

ADDITIVE Causes the new text to be appended to existing text. REPLACE MEMO without the ADDITIVE option causes *Visual* dBASE to overwrite any text currently in the memo field.

Description

Use the REPLACE MEMO...FROM command to insert a text file into a memo field. You can insert one text file to each memo field of each record in a table.

While dBASE memo fields may contain types of information other than text, binary fields are recommended for storing images, sound, and other user-defined binary type information. Use OLE fields for linking to OLE documents from other Windows applications.

Example

In this example, the Customer file is scanned. If there is a file of the form CUST????.DOC, where ???? is the 4 character company code, then REPLACE MEMO...FROM copies the file into the Notes memo field:

```
CLOSE ALL
USE Customer
SCAN
  FN="Cust"+Compcode+".Doc"
  IF FILE("&FN")
    REPLACE MEMO Notes FROM &FN
  ELSE
    BLANK FIELD Notes
  ENDIF
ENDSCAN
```

Portability

Not supported in dBASE IV or dBASE III PLUS.

See Also

COPY MEMO, REPLACE BINARY, REPLACE MEMO, REPLACE OLE

REPLACE OLE

Fields and records

Inserts an OLE document into an OLE field.

Syntax

```
REPLACE OLE <OLE field name>
FROM <filename> | ? | <filename skeleton>
[LINK]
```

<OLE field name> The field where an OLE document is inserted.

FROM <file name> | ? | <filename skeleton> The file that identifies an OLE document, including its extension. If you specify a file without including its path, dBASE looks for the file in the current directory, then in the path you specify with SET PATH. The ? and <filename skeleton> options display a dialog box in which you specify the file.

LINK LINK provides a pointer to the OLE document. By default, dBASE embeds the OLE document in the specified memo field.

Description

Use REPLACE OLE to insert the contents of an OLE document into an OLE field. You can either embed the actual OLE document in an OLE field (the default) or access the OLE document by linking it to the OLE field.

If you link the OLE document, the OLE field contains only a reference to the OLE document. As long as the OLE document remains in the same location, the OLE field displays the most current version of the document.

If you embed the OLE document, the OLE field contains a copy of the document. There are no links between the field and the OLE document: therefore, any changes to the original version of the OLE document are not reflected in the embedded document.

Example

The following example demonstrates REPLACE OLE. It adds a new record to the Pictures table. The Name field is replaced with "AirBrInd" and the bitmap file AIRBRLND.BMP is placed in the BitMapOle field:

```
USE Pictures
APPEND BLANK
REPLACE Name WITH "AirBrInd"
REPLACE OLE BitMapOle FROM "AirBrInd.BMP"
```

If you have AIRBRLND.BMP you can examine this record with EDIT or BROWSE and click on the BitMapOle field.

Portability

Not supported in dBASE IV or dBASE III PLUS.

See Also

CLASS OLE, DEFINE

REPLICATE()**String data**

Returns a string repeated a specified number of times.

Syntax

REPLICATE(<expC> | <memo field>, <expN>)

<expC> | <memo field> The string or memo field to repeat.

<expN> The number of times to repeat the string or memo field.

Description

REPLICATE() returns a character string composed of a character expression or memo field repeated a specified number of times. dBASE displays an error if the resulting string exceeds 32766 characters. Therefore, the number of repeats you specify must be less than 32766 divided by the number of characters in the character expression or memo field.

If the character expression is an empty string or the memo field is empty, REPLICATE() returns an empty string. If the number of repeats you specify for <expN> is 0, REPLICATE() returns an empty string. If <expN> is less than 0, dBASE displays an error.

To repeat space characters, use SPACE().

Example

The following example uses REPLICATE() to create different strings of characters:

```
? REPLICATE("+",10)           && Returns "+++++++"
? REPLICATE("",10)           && Returns ""
? REPLICATE("-",10)          && Returns "-----"
? REPLICATE("dBASE",2)       && Returns "dBASEdBASE"
a_s = REPLICATE("A",32766)
? LEN(a_s)                   && Returns 32766
? REPLICATE("dBASE for Windows!",1820)
* 32766/18 characters in the string = 1820 times
```

Portability

The <memo field> argument isn't supported in dBASE IV or dBASE III PLUS. Both dBASE IV and dBASE III PLUS limit the return value of REPLICATE() to 254 characters.

See Also

SPACE()

REPORT FORM

Input/Output

Generates and displays or prints a report, using the report format stored in a specified report file and information derived from records in the current table.

Syntax

```
REPORT FORM <filename 1> | ? | <filename skeleton 1>
[<scope>] [FOR <condition 1>] [WHILE <condition 2>]
[CROSSTAB]
[HEADING <expC>]
[NOEJECT]
[PLAIN]
[SUMMARY]
[TO FILE <filename 2> | ? | <filename skeleton 2>] [TO PRINTER]
```

<filename 1> | ? | <filename skeleton> The report format file to use. The ? and <filename skeleton> options display a dialog box from which you can select a file. If you specify a file without including its path, dBASE looks for the file in the current directory, then in the path you specify with SET PATH. If you specify a file without including its extension, dBASE looks for .RPT, .FRG, or .FRM, in that order. If you specify CROSSTAB, dBASE looks for .RPC, .FRG, or .FRM, in that order.

<scope> The number of records to derive the report from. RECORD <n> identifies a single record by its record number. NEXT <n> identifies n records, beginning with the current record. ALL specifies all records. REST specifies all records from the current record to the end of the file.

FOR <condition 1>

WHILE <condition 2> Determines which records are affected by REPORT FORM. FOR restricts REPORT FORM to records that meet <condition 1>. WHILE starts processing with the current record and continues with each subsequent record as long as <condition 2> is true.

CROSSTAB Specifies that the report was created with the Cross-Tab dialog box.

HEADING <expC> Adds a character expression, <expC>, as the heading of each page. HEADING has no effect if used with PLAIN.

NOEJECT Prevents a page feed before printing begins.

PLAIN Suppresses page numbers and dates on the pages of the report. Any title appears only on the first page.

SUMMARY Includes only the totals of groups and subtotals of subgroups, excluding the individual contents of records within each group and subgroup. The report format defines groups and subgroups.

TO FILE <filename 2> | ? | <filename skeleton 2> Directs output to the text file <filename 2>. By default, dBASE assigns a .TXT extension to <filename 2> and saves the file in the current directory. The ? and <filename skeleton> options display a dialog box in which you specify the name of the target file and the directory to save it in.

TO PRINTER Directs output to the printer.

R

Description

Use REPORT FORM to print or display reports in a format that you've defined in the Report Designer using CREATE REPORT or MODIFY REPORT. For information about using the Report Designer, see the Crystal Reports documentation. If you don't specify a <scope>, WHILE <condition 1>, or FOR <condition 2> option, REPORT FORM prints the report specifications for each record in record number or index order.

When printing or displaying a report that includes groups of data or group subtotals, the current table must either be in sorted order or its master index must be in use. The sorted file or index must be arranged according to the value of the field on which the data is grouped.

REPORT FORM without the TO FILE or TO PRINTER options displays the report in the Command window or current user-defined window.

Example

This example opens the Company database and then generates a report using the Comprep1 report definition:

```
CLOSE DATABASE
USE Company
REPORT FORM Comprep1 TO PRINT
```

Portability

The <filename skeleton> option is not supported in dBASE IV or dBASE III PLUS.

See Also

CREATE REPORT

RESOURCE()

Windows programming

Returns a character string from a DLL file.

Syntax

RESOURCE(<resource id>, <DLL filename expC>)

<resource id> A numeric value that identifies the character string resource.

<DLL filename expC> The name of the DLL file.

Description

Use RESOURCE() to generate a character string from a resource in a DLL file. The character string must be less than 32K; a character string longer than this is truncated.

RESOURCE() is often useful for internationalizing applications without changing program code. For example, you can store in a DLL file all character strings that might need translation from one language to another, and your application can retrieve them at run time with the RESOURCE() function. To modify the application for another language, translate the strings and store them in a new DLL file, in the same order and

with the same resource IDs as their counterparts in the original DLL file. Then substitute the new DLL for the original one. For more information on DLL files, see EXTERN, LOAD DLL, and Chapter 25 in the *Programmer's Guide*.

Portability

Not supported in dBASE IV or in dBASE III PLUS.

Example

This example shows how RESOURCE() can be used to change languages for international development:

```
#define ENGLISH 1
#define SPANISH 2
ApLanguage = SPANISH
MyGreeting = RESOURCE(SPANISH,"greeting.dll")
CLEAR
? MyGreeting
```

See Also

EXTERN, LOAD DLL, RELEASE DLL

RESTORE

Memory variables

Copies the memory variables stored in the specified disk file to active memory.

Syntax

```
RESTORE FROM <filename> | ? | <filename skeleton>
[ADDITIVE]
```

<filename> | ? | <filename skeleton> The file of memory variables to restore. RESTORE FROM ? and RESTORE FROM <filename skeleton> display a dialog box, from which you can select a file. If you specify a file without including its path, dBASE looks for the file in the current directory, then in the path you specify with SET PATH. If you specify a file without including its extension, dBASE assumes .MEM.

ADDITIVE Preserves existing memory variables when RESTORE is executed.

Description

Use RESTORE with SAVE to retrieve and store important memory variables. By default, all private variables are cleared at the end of execution of the program file that created them, while all public variables are cleared when you exit dBASE. To preserve these values for future use, store them in a memory file by using SAVE. You can then retrieve these values later by using RESTORE.

Without the ADDITIVE option, RESTORE clears all existing user memory variables before returning to active memory the variables stored in a memory file. Use ADDITIVE when you want to restore a set of variables while retaining those already in memory.

Note If you use ADDITIVE and a restored variable has the same name as an existing variable, the restored variable will replace the existing one.

R

If you issue RESTORE in the Command window, dBASE makes all restored variables public. When dBASE encounters RESTORE in a program file, it makes all restored variables private to that program.

Example

The following example stores Company name to the variable Start when the user exits a BROWSE. The value in Start is then saved to an external .MEM file with the SAVE command. The next time the program is run, RESTORE FROM is used to recover the variable Start, and SEEK positions the record pointer to the appropriate Company so BROWSE can be resumed at the former location:

```
SET TALK OFF
SET SAFETY OFF
USE Clients EXCLUSIVE
INDEX ON Company TAG Company
IF FILE("Lst_Look.MEM")
    RESTORE FROM Lst_Look      && Recover Start
    SEEK Start
    BROWSE
ELSE
    GO TOP
    BROWSE
ENDIF
STORE Company TO Start
SAVE TO Lst_Look.MEM
CLOSE ALL
SET TALK ON
SET SAFETY OFF
```

Portability

The ? and <filename skeleton> arguments are not supported in dBASE IV or dBASE III PLUS.

See Also

CLEAR MEMORY, RELEASE, SAVE, SET PATH, STORE

RESTORE IMAGE

Objects

Displays an image stored in a file or a binary field.

Syntax

```
RESTORE IMAGE FROM
<filename> | ? | <filename skeleton> | BINARY <binary field>
[TIMEOUT <expN>]
[TO PRINTER]
[[TYPE] PCX | TIF[F] | ICO | WMF | EPS]
```

FROM <filename> | ? | <filename skeleton> | BINARY <binary field> Identifies the file or binary field to restore the image from. RESTORE IMAGE FROM ? and RESTORE IMAGE FROM <filename skeleton> display the Open Source File dialog box, which lets the user

select a file. *<filename>* is the name of an image file; RESTORE IMAGE assumes a .BMP extension and file type unless you specify otherwise. If you specify a file without including its path, dBASE looks for the file in the current directory, then in the path you specify with SET PATH. RESTORE IMAGE FROM BINARY *<binary field>* displays the image stored in a binary field. You store an image in a binary field with the REPLACE BINARY command.

TIMEOUT *<expN>* Specifies the number of seconds the image is displayed onscreen.

TO PRINTER Sends the image to the printer as well as to the screen.

[TYPE] PCX Specifies an image stored in PCX format, and assumes a .PCX file-name extension if none is given. The word TYPE is optional.

Description

Use RESTORE IMAGE to display a graphic image that was generated and saved in bitmap or PCX format. The image is displayed in a window.

Notes Your computer must have a graphics adapter to display an image.

To print an image with the TO PRINTER option, you must have a printer that can process and print graphic data.

Example

The following example defines a form and list box for selection of an aircraft model and uses RESTORE IMAGE to display a graphic from the memo field Image of the selected record:

```
CLOSE ALL
SET TALK OFF
USE Aircrdb ORDER Aircraft IN SELECT()
SELECT Aircrdb
DEFINE FORM AC
    PROPERTY
        Top 5,
        Left 2,
        Height 13,
        Width 30,
        Text "Aircraft",
        Sizeable .T.,
        OnSelection Photo
DEFINE LISTBOX Model OF AC
    PROPERTY
        Top 2,
        Left 10,
        Height 7,
        Width 18,
        DataSource "FIELD Aircrdb->Aircraft"
OPEN FORM AC

FUNCTION Photo
RESTORE IMAGE FROM BINARY Image
RETURN .T.
```

Portability

Not supported in dBASE IV or dBASE III PLUS.

See Also

DEFINE, REPLACE BINARY

RESTORE SCREEN

Input/Output

Redisplays in the results pane of the Command window its previous contents saved with SAVE SCREEN. This command is supported primarily for compatibility with dBASE IV.

For complete syntax information on CLEAR SCREENS, see online Help.

RESTORE WINDOW

dBASE IV windows

Extracts window definitions from a window file and loads them in memory. This command is supported primarily for compatibility with dBASE IV. In *Visual dBASE*, run the program, procedure, or library file that contains stored form definitions.

For complete syntax information on RESTORE WINDOW, see online Help. For information about defining forms, see the Forms chapters in the *User's Guide*.

RESUME

Error handling and debugging

Restarts program execution at the command line following the one at which program execution was suspended.

Syntax

RESUME

Description

RESUME causes dBASE to resume execution of a program that is suspended. You can suspend program execution by issuing SUSPEND. If you have not assigned a value to ON ERROR, you can also choose to suspend a program when an error occurs.

To restart program execution, enter RESUME in the Command window. The program file resumes execution at the line immediately following the line that caused it to become suspended. If you want to re-execute the line that caused an error, perhaps because you fixed the condition that caused the error, retype the program line at the command line before issuing RESUME.

Example

See SUSPEND for an example of how to use RESUME after a SUSPEND command has been executed in a program.

See Also

CANCEL, ON ERROR, RETRY, RUN, SUSPEND

RETRY**Error handling and debugging**

Returns control from a subroutine to the command line of the calling routine or Command window that called the subroutine.

Syntax

RETRY

Description

Use RETRY to re-execute a command—for example, one that resulted in an error. RETRY returns program control to the calling command. RETRY clears the memory variables created by the subroutine.

RETRY is valid only in program files.

You can use RETRY with ON ERROR to give the user more chances to resolve an error condition. Using RETRY with ON ERROR resets ERROR() to zero.

Example

The following example uses the Recover procedure when an error is detected with ON ERROR. If the Clients table is already open in another work area when the USE command is executed, an error is returned. The Recover procedure uses CLOSE DATABASES to insure that all tables are closed and RETRY returns program flow to the USE command:

```
ON ERROR DO Recover
USE Clients EXCLUSIVE
BROWSE
ON ERROR
CLOSE DATABASES

PROCEDURE Recover
WAIT "An error has occurred.;
    Press any key to retry.."
CLOSE DATABASES
RETRY
```

R

See Also

ERROR(), MESSAGE(), ON ERROR, RESUME, RETURN

Ends execution of a program, procedure, or user-defined function (UDF), returning control to a calling routine—program, procedure, or UDF—or to the Command window.

Syntax

RETURN

[<return exp> | TO MASTER | TO <routine name>]

<return exp> The value a procedure UDF returns to the calling routine or the Command window. RETURN <return exp> must be the last line in a procedure or UDF definition; <return exp> is required for UDFs.

TO MASTER Returns control to the highest-level calling routine or to the Command window. As one routine calls another, each goes onto the *call stack*, a list of pending routines and subroutines. TO MASTER returns to the bottom routine in the stack (the master procedure), terminates all subroutines in the call stack, and releases all nonpublic variables at each level.

If RETURN TO MASTER executes in the master procedure, control returns to the Command window. If the call stack started from a suspended prompt in the Command window, control returns to the suspended prompt.

TO <routine name> Returns control to the specified calling routine, terminating all intervening subroutines on the call stack and releasing those subroutines' non-public variables. (See the previous description of TO MASTER for an explanation of *call stack*.)

dBASE handles special cases as follows:

- If <routine name> is the name of the current routine, a regular RETURN occurs.
- If <routine name> occurs more than once in the call stack, control passes to the routine by that name that is closest in the stack (the one that made the most recent call).
- If <routine name> is *master*, the TO MASTER option takes precedence and returns control to the highest-level calling routine.
- If <routine name> isn't in the call stack, RETURN causes a run-time error.

Description

When a procedure ends, dBASE automatically returns to the calling routine. You can use RETURN in procedures to return from a point in the procedure other than its end. You must use RETURN at the end of UDFs to return the result (the return value) of the UDF.

Use RETURN in programs to restore control to calling routines or to the Command window. Command processing generally resumes as follows:

- If the routine is called with a DO statement, command processing continues at the line following the DO statement.
- If the routine is a UDF called directly as a command or as part of an expression (not with DO), command processing continues in the statement containing the call.

- If you use the TO MASTER or TO *<routine name>* options of the RETURN command, command processing continues with the next command line after the last one executed in MASTER or *<routine name>*.

RETURN does all of the following:

- Releases all non-public memory variables defined in the routine returned from (and in any intervening routines on the call stack), but doesn't affect public variables or the values of static variables in that routine
- Sets the ERROR() function to 0
- Changes *scope*—cognition of which private, local, and static system variables are in effect and what their values are—to that of the routine returned to, restoring previous values of private, local, and static system variables for that routine

You can also use CANCEL to end execution of a routine, but CANCEL always returns control to the Command window.

Example

The following examples show various uses RETURN. In GetOut you can RETURN to the main program or to an open procedure. The RETURN in IsReady() shows a value being returned.

```
DO PrintTo With "LPT1:"
DO GetOut With .t.
DO GetOut With .f.
Answer= IsReady()
```

```
PROCEDURE PrintTo
PARAMETER OutPutDev
* ...
RETURN
```

```
PROCEDURE GetOut
Parameter Done
IF Done
    RETURN TO MASTER
ELSE
    RETURN TO ReportMenu
ENDIF
```

```
FUNCTION IsReady
RETURN .T.
```

R

Portability

The *<return exp>* and TO *<routine name>* options aren't supported in dBASE III PLUS.

See Also

CANCEL, DO, ERROR(), FUNCTION, LOCAL, PRIVATE, PROCEDURE, PUBLIC, QUIT, RETRY, STATIC

RIGHT()

Returns characters from the end of a character string or memo field.

Syntax

RIGHT(<expC> | <memo field>, <expN>)

<expC> | <memo field> The string or memo field to extract characters from.

<expN> The number of characters to extract from the string or memo field.

Description

Starting with the last character of a character expression or memo field, RIGHT() returns a specified number of characters. RIGHT() returns a maximum of 32766 characters, the maximum length of a string.

If the number of characters you specify for <expN> is greater than the number of characters in the specified string or memo field, RIGHT() returns the string as is, without adding space characters to achieve the specified length. If <expN> is less than or equal to zero, RIGHT() returns an empty string.

When RIGHT() returns characters from a memo field, it counts two characters for each carriage-return and linefeed combination (CR/LF).

Example

The following example uses RIGHT() to return a portion of a text string, starting from the right end of the string:

? RIGHT("dBASE",1)	&& Returns "E"
? RIGHT("dBASE",3)	&& Returns "ASE"
? RIGHT("dBASE",9)	&& Returns "dBASE"
? RIGHT("dBASE",0)	&& Returns ""

The next example uses RIGHT() to order a file by the last three characters of the Zip_P_Code field.

```
USE Clients EXCLUSIVE
INDEX ON RIGHT(Zip_P_Code,3) TAG Zipdex
LIST FIELDS Company, Zip
CLOSE ALL
```

Portability

The <memo field> argument isn't supported in dBASE IV or dBASE III PLUS. Both dBASE IV and dBASE III PLUS limit the return value of RIGHT() to 254 characters.

See Also

AT(), LEFT(), RAT(), SUBSTR()

RLOCK()

Shared data

Locks the current record or a specified list of records in the current or specified alias table, and returns .T. if successful.

Syntax

```
RLOCK([<list expC>] | [<bookmark list expC>]  
[, <alias>])
```

<list expC> The list of record numbers to lock, separated by commas.

<bookmark list expC> The list of bookmarks (record indicators) returned by BOOKMARK() specifying a record in a non-dBASE table, such as a Paradox table, that doesn't have natural record-order record numbers. Separate bookmarks with commas.

<alias> A work area number (1 through 225), letter (A through J), or alias name. The work area letter or alias name must be enclosed in quotes. If you don't include <alias>, RLOCK() acts on the current table.

You don't have to specify record numbers or bookmarks if you want to specify a value for <alias>. However, if you have specified record numbers or bookmarks, you must precede <alias> with a comma (,).

Description

Use RLOCK() to lock the current record or a list of records in the current or an alias table. If you don't pass RLOCK() any arguments, it locks the current record in the current table. If you pass only <alias> to RLOCK(), it locks the current record in the alias table. If RLOCK() is successful in locking *all* the records you specify, it returns .T. You can lock up to 100 records in each table open at your workstation with RLOCK().

You can view and update a record you lock with RLOCK(). Other users can view this record but can't update it. When you lock a record with RLOCK(), it remains locked until you do one of the following:

- Issue UNLOCK
- Press *Ctrl+L* (the lock/unlock toggle command) in a Browse or Edit window when the record pointer is on a record you locked with RLOCK()
- Close the table

RLOCK() is similar to FLOCK(), except FLOCK() locks an entire table. Use FLOCK(), therefore, when you need to have sole access to an entire table or related tables—for example, when you need to update multiple tables related by a common key—or when you want to update more than 100 records at a time.

All commands that change table data cause dBASE to attempt to execute an automatic record or file lock. If dBASE fails to execute an automatic record or file lock, it returns an error. You might want to use RLOCK() for event trapping, testing for its return value rather than for an error condition.

RLOCK() can't lock the records you specify when any of the following conditions exist:

RLOCK()

- Another user has locked, explicitly or automatically, the current record or one of the records in <list expC> or <bookmark list expC>.
- Another user has locked, explicitly or automatically, the current table or the specified alias table.

By default, when RLOCK() can't immediately lock the records you specify, dBASE prompts you to retry locking the records or to cancel further retries. Use SET REPROCESS to specify the number of retries. If you choose to cancel, RLOCK() returns .F.

When you set a relation to a *parent table* with SET RELATION and then lock a record in the table with RLOCK(), dBASE attempts to lock all *child records* in *child tables*. For more information on relating tables, see SET RELATION.

RLOCK() is equivalent to LOCK().

Example

This example loops until it can lock the 10th record with RLOCK() or until the user decides to stop trying. If successful, a subroutine, CompMod, is called to update the record:

```
RecordWasRead = .t.
SET REPROCESS TO 20  && Try to lock 20 times
USE Company
GO 10
Again = .t.
DO WHILE Again
  IF RLOCK()          && Can dBASE Lock the file?
    DO CompMod        && Update this record
    Again = .f.
    UNLOCK            && Allow other users to change
                     && the file.
  ELSE                && RLOCK() returns .F.
    CLEAR
    Wait "Record lock failed. Try again? (Y/N) ";
      to mRetry
    IF UPPER(mRetry)="N"
      Again = .f.
    ENDIF
    RecordWasRead = .f.
  ENDIF
ENDDO
USE
SET REPROCESS TO 0  && The default
```

LOCK() can lock more than one record:

```
USE Games ALIAS Fun
* ...
Pythagoras= LOCK("8,15,17","Fun")
```

Portability

Not supported in dBASE III PLUS. The <bookmark list expC> option isn't supported in dBASE IV.

See Also

FLOCK(), SET LOCK, SET RELATION, SET REPROCESS, UNLOCK

ROLLBACK()

Shared data

Ends a transaction initiated by BEGINTRANS() without saving any changes to the open files. Returns .T. if the data was rolled back successfully.

Syntax

ROLLBACK([<database name expC>])

<database name expC> The name of the database in which to cancel the transaction.

- If you began the transaction with BEGINTRANS(<database name expC>), you must issue ROLLBACK(<database name expC>). If instead you issue ROLLBACK(), dBASE ignores the ROLLBACK() statement.
- If you began the transaction with BEGINTRANS(), <database name expC> is an optional ROLLBACK() argument. If you include it, it must refer to the same database as the SET DATABASE TO statement that preceded BEGINTRANS().

Description

Use ROLLBACK() to end the open transaction and restore any open files to the state they were in when BEGINTRANS() was issued. To end a transaction and write changes to the files, use COMMIT(). For more information on transactions, see BEGINTRANS().

Example

The following example begins a transaction with BEGINTRANS(). It opens a multi-user version of Company.dbf and attempts to set values in the Ytd_Sales field to 0. ON ERROR detects any error which might occur. In particular, it will detect if another user has locked any record in Company.dbf. If an error occurs, ROLLBACK() resets all values. Otherwise COMMIT() writes the changes to disk:

```
CLOSE ALL
SET PROCEDURE TO PROGRAM(1) ADDITIVE
SET EXCLUSIVE OFF

BEGINTRANS()

TransErr=.f.
ON ERROR DO TransErr    && Activates ON ERROR trap

USE L:\MultiUse\Company
REPLACE ALL Ytd_Sales WITH 0
ON ERROR                && Disables ON ERROR

IF TransErr
? "Rollback"
ROLLBACK()              && restore data
ELSE
? "Commit"
```

R

ROUND()

```
        COMMIT()                && save changes
    ENDIF

    PROC TransErr
    WAIT "Warning: Transaction Fails"
    TransErr=.t.
```

Portability

Not supported in dBASE IV or dBASE III PLUS. ROLLBACK() replaces the ROLLBACK command in dBASE IV.

See Also

BEGINTRANS(), COMMIT(), SET EXCLUSIVE

ROUND()

Numeric data

Returns a specified number rounded to a specified number of decimal places.

Syntax

ROUND(<expN 1>, <expN 2>)

<expN 1> The numeric or float number to round.

<expN 2> If <expN 2> is positive, the number of decimal places to round <expN 1> to. If <expN 2> is negative, whether to round <expN 1> to the nearest tenth, hundredth, thousandth, and so on.

Description

Use ROUND() to round a number to a specified number of decimal places or to a specified tenth, hundredth, thousandth value, and so forth. Use ROUND() with SET DECIMALS to round a number *and* remove trailing zeros.

If the digit in position <expN 2> + 1 is between 0 and 4 inclusive, <expN 1> (with <expN 2> decimal places) remains the same; if the digit in position <expN 2> + 1 is between 5 and 9 inclusive, the digit in position <expN 2> is increased by 1.

Use 0 as <expN 2> to round a number to the nearest whole number. Using -1 rounds a number to the nearest tenth; rounding to a -2 rounds a number to the nearest hundredth; and so on. For example, ROUND(14932,-2) returns 14900 and ROUND(14932,-3) returns 15000.

See the table in the description of INT() that compares INT(), FLOOR(), CEILING(), and ROUND().

Example

The following example demonstrates the use of ROUND() as the decimal place parameter of the function is incremented:

```

SET DECIMALS TO 6
SET TALK OFF
x = 14.746321
? "    x = 14.746321"
FOR y = -5 TO 5 STEP 1
    ? "If y = " + STR(y,2,0)
    ?? "    ROUND(x,y) returns "
    ?? ROUND(x,y)
NEXT
SET TALK ON

```

See Also

ABS(), CEILING(), FLOOR(), INT()

ROW()

Input/Output

Returns the number of the current row position in the Results pane of the Command window or the current dBASE IV window. This command is supported primarily for compatibility with dBASE IV. In *Visual* dBASE, use the Top property of a class to determine its vertical position on a form.

For complete syntax information on ROW(), see online Help. For more information about working with *Visual* dBASE forms, see the Forms chapters in the *User's Guide*.

RTOD()

Numeric data

Returns the degree value of an angle measured in radians.

Syntax

RTOD(<expN>)

<expN> A negative or positive integer or float that is the size of the angle in radians.

Description

RTOD() converts the measurement of an angle from radians to degrees. RTOD() returns a float.

To convert radians to degrees, dBASE

- Multiplies the number of radians by 180
- Divides the result by PI()
- Returns the quotient

An angle of pi radians is equivalent to 180 degrees.

Use RTOD() with the trigonometric functions ACOS(), ASIN(), ATAN(), and ATN2() to convert the radian return values of these functions to degrees. For example, if the default number of decimal places is 2, ATAN(1) returns the value of the angle in radians, 0.79, while RTOD(ATAN(1)) returns the value of the angle in degrees, 45.00.

R

RTRIM()

Use SET DECIMALS to set the number of decimal places RTOD() displays.

Example

The following example uses RTOD() to convert radians to degrees and returns degrees. One pi radian is equal to 180 degrees:

```
? RTOD(pi())           && Returns 180
```

Portability

Not supported in dBASE III PLUS.

See Also

ACOS(), ASIN(), ATAN(), ATN2(), COS(), DTOR(), PI(), SET DECIMALS, SIN(), TAN()

RTRIM()

String data

Returns a string with no trailing space characters.

Syntax

RTRIM(<expC> | <memo field>)

<expC> | <memo field> The string or memo field to remove the trailing space characters from.

Description

RTRIM() is identical to TRIM(). See the description of TRIM() for more information.

Example

See the example of TRIM(); substitute RTRIM() for TRIM()

Portability

The <memo field> argument isn't supported in dBASE IV or dBASE III PLUS. Both dBASE IV and dBASE III PLUS limit the return value of RTRIM() to 254 characters.

See Also

LEFT(), LTRIM(), RIGHT(), TRIM()

RUN

Disk and file utilities

Executes a single DOS command or program from within dBASE.

Syntax

`RUN <DOS command>`

<DOS command> A command recognized by your DOS operating system.

Description

Use RUN to execute a single DOS command or program file without exiting dBASE. Enter commands and file names exactly as you would when working in DOS; do not enclose them in quotes. ! is equivalent to RUN.

When execution of the command or program file is completed, control returns to the Command window or to the program, procedure, or user-defined function (UDF) that executed RUN.

When you issue commands that run in a DOS window, you can use the Windows PIF Editor to customize the settings in DBASEWIN.PIF. For example, if you don't want the DOS window to close automatically after the command is completed, you can clear the Close Window on Exit checkbox in DBASEWIN.PIF. For more information about .PIF files, see your Windows documentation.

To execute multiple DOS commands interactively without exiting dBASE, use the DOS command. To execute a Windows application without exiting dBASE, use RUN().

If you want to change the current directory being used by dBASE, use CD instead of RUN. Issuing RUN CD will change the directory only in the DOS window; the directory change will not be reflected in the dBASE system.

Example

RUN must have an argument:

```
RUN dir
* Performs the DOS directory command
RUN && gives a syntax error
```

See Also

CD, DOS, HOME(), RUN(), SET DIRECTORY, SET PATH

R

RUN()

Disk and file utilities

Executes a single DOS command, a DOS application, or a Windows application from within dBASE, and returns an exit code from the DOS command interpreter or the Windows application.

Syntax

`RUN([<expL1>,<DOS command expC> [<expL2>]])`

<expl1> Determines whether RUN() runs a Windows program (value of .T.) or a DOS program (value of .F.). If you do not include *<expl1>*, dBASE assumes a value of .F.

<DOS command expC> A command recognized by your DOS operating system. Unlike the RUN command, this command must be *<expC>*.

<expl2> Visual dBASE ignores this parameter; it is included for backward compatibility with dBASE IV.

Description

Use RUN() to execute a single DOS command or an external (DOS or Windows) application without exiting dBASE. To execute multiple DOS commands interactively without exiting dBASE, use the DOS command.

When execution of the command or program file is completed, control returns to the Command window or to the program, procedure, or user-defined function (UDF) that called RUN(), and an exit code is returned.

For RUN() to access DOS (as when *<expl1>* = .F. or when *<expl1>* is omitted), dBASE loads COMMAND.COM using the DBASEWIN.PIF in the directory where DBASEWIN.EXE is located. (To determine that directory, use HOME().) You can use the Windows PIF Editor to customize the settings in DBASEWIN.PIF. For example, if you don't want the DOS window to close automatically after the command is completed, you can clear the Close Window on Exit checkbox in DBASEWIN.PIF. For information about .PIF files, see your Windows documentation.

If a DOS command or application is executed, the exit code returned by RUN() is 0 if the command or the application executes successfully. When RUN() executes Windows applications, the value returned is the instance handle of the application (if execution was successful) or an error number (if execution was unsuccessful). Each instance handle is a 16-bit integer that identifies an active Windows application, and each error number identifies a Windows error condition.

Example

This command attempts to run the Brief Editor in DOS. If it succeeds then Result =0; if not, Result will be nonzero.

```
Result=RUN(.f., "B")
```

These commands attempt to run Windows programs:

```
Result=RUN(.t., "winfile.exe")
* e.g. returns 10798
Result=RUN(.t., "nosuch.exe")
* e.g. returns 2
```

Portability

Not supported in dBASE III PLUS. In dBASE IV, *<expl1>* determines whether the operating system's command interpreter is loaded, and *<expl2>* determines whether all available extended memory is released prior to running *<DOS command>*.

See Also

DOS, HOME(), RUN

SAVE**Memory variables**

Stores all or specified memory variables to a designated memory file.

Syntax

SAVE TO <filename> | ? | <filename skeleton>

[ALL]

[LIKE <memvar skeleton 1>]

[EXCEPT <memvar skeleton 2>]

TO <filename> | ? | <filename skeleton> Directs the memory variable output to be saved to the target file <filename>. By default, dBASE assigns a .MEM extension to <filename> and saves the file in the current directory. The ? and <filename skeleton> options display a dialog box in which you specify the name of the target file and the directory to save it in.

ALL Stores all memory variables to the memory file. If you issue SAVE TO <filename> with no options, dBASE also saves all memory variables to the memory file.

LIKE <memvar skeleton 1> Stores in the target file the memory variables whose names are like the memory variable skeleton you specify for <memvar skeleton 1>. Use characters of the variable names and the wildcards * and ? to create <memvar skeleton 1>.

EXCEPT <memvar skeleton 2> Stores in the target file all memory variables except those whose names are like the memory variable skeleton you specify for <memvar skeleton 2>. Use characters of the variable names and the wildcards * and ? to create <memvar skeleton 2>.

Description

Use SAVE with RESTORE to store and retrieve important memory variables. Private variables are cleared at the end of the program file that created them, while public variables are cleared when you exit dBASE. To preserve these values for future use, store them in a memory file with SAVE. Use RESTORE to retrieve them.

If SET SAFETY is ON and a file exists with the same name as the target file, dBASE displays a dialog box asking if you want to overwrite the file. If SET SAFETY is OFF, any existing file with the same name is overwritten without warning.

Note SAVE does not save object reference, function pointer, or system memory variables.

S**Example**

See RESTORE for an example of using SAVE.

Portability

The ? and | <filename skeleton> options are not supported in dBASE IV or dBASE III PLUS. dBASE IV and dBASE III PLUS do not support both LIKE and EXCEPT in the same SAVE statement.

In dBASE IV and dBASE III PLUS, you must specify ALL if you want to specify LIKE or EXCEPT. In *Visual* dBASE, a statement such as SAVE TO memfile LIKE mvar* is acceptable; the ALL option is assumed.

See Also

RELEASE, RESTORE, SET SAFETY, STORE

SAVE SCREEN

Input/Output

Saves the contents of the results pane of the Command window. This command is supported primarily for compatibility with dBASE IV. In *Visual* dBASE, form definitions are saved in program, procedure, or library files.

For complete syntax information on SAVE SCREEN, see online Help. For information about defining forms, see the Forms chapters in the *User's Guide*.

SAVE WINDOW

dBASE IV windows

Saves window definitions to a window file. This command is supported primarily for compatibility with dBASE IV. In *Visual* dBASE, form definitions are saved in program, procedure, or library files.

For complete syntax information on SAVE WINDOW, see online Help. For information about defining forms, see the Forms chapters in the *User's Guide*.

SCAN

Programs

Steps through each record in the current table, executing specified statements on each record that meets specified conditions.

Syntax

```
SCAN
    [<scope>] [FOR <condition 1>] [WHILE <condition 2>]
    [<statements>]
    [LOOP]
    [EXIT]
ENDSCAN
```

<scope> The number of records to scan. RECORD <n> identifies a single record by its record number. NEXT <n> identifies *n* records, beginning with the current record. ALL specifies all records. REST specifies all records from the current record to the end of the table.

FOR <condition 1>

WHILE <condition 2> Determines which records are affected by SCAN. FOR restricts SCAN to records that meet <condition 1>. WHILE starts processing with the current record and continues with each subsequent record as long as <condition 2> is true.

<statements> Program lines consisting of any combination of commands, functions, user-defined functions (UDFs), and LOOP and EXIT options. Because SCAN processes a table record by record, these statements are likely to include table commands and functions such as ISBLANK() and REPLACE.

LOOP Moves the record pointer ahead one record and returns program control to the beginning of the loop without performing the statements that follow LOOP and precede ENDSCAN.

EXIT Transfers program control out of the SCAN loop to the statement following ENDSCAN without executing the statements that follow EXIT and precede ENDSCAN and without processing any more records.

ENDSCAN A required command that marks the end of the SCAN loop. When dBASE encounters ENDSCAN, it moves the record pointer ahead one record and returns program control to the beginning of the loop.

Description

Use SCAN to process the current table record by record, starting with the first record in the table or master index or the first record that meets a FOR condition. For each record that falls within *<scope>* or meets a FOR condition or a WHILE condition or any combination of the three, dBASE executes all the statements after SCAN until it encounters LOOP, EXIT, or ENDSCAN. The SCAN loop continues to the end of the table (or the last record in the index) unless the *<scope>*, the FOR or WHILE condition, or the EXIT option ends the processing earlier. The record pointer remains where it is when processing ends.

At the end of each loop, dBASE automatically moves the record pointer forward one record in the table before returning to the beginning of the loop; therefore, don't include a SKIP command.

You can nest loops and other structures, including other SCAN loops, in a SCAN loop. Each ENDSCAN in a nested group of SCANS moves the record pointer one record. The nested code must account for this record pointer movement, or code might execute on records you didn't intend it to.

SCAN works like a DO WHILE .NOT. EOF()...SKIP...ENDDO construct; however, with SCAN you can specify conditions with FOR, WHILE, and *<scope>*. SCAN also requires fewer lines of code than DO WHILE.

When using SCAN with an indexed table, don't change the value of a field that is (or is part of) the master index key. When you change the value of such a field, dBASE repositions the record in the index file, which might cause unintended results. For example, if you change a key field that causes its record to move to the end of the index, that record might have the SCAN...ENDSCAN statements executed on it a second time.

If you change work areas within a SCAN loop, select the work area containing the table being scanned before control passes back to the first statement in the SCAN loop.

Example

The following example uses SCAN to step through a table and calls an invoicing procedure for those records that have a positive value in field Startbal:

SECONDS()

```
CLEAR
USE Clients
SCAN FOR Startbal > 0
    DO Invoice WITH Company, Startbal, Baldate
ENDSCAN

PROCEDURE Invoice
PARAMETERS Company, Startbal, Baldate
? Company, Startbal, Baldate
RETURN
```

The following code is equivalent to the SCAN loop used in the previous example but uses the DO WHILE and IF/ENDIF commands:

```
SET TALK OFF
USE Clients
DO WHILE .NOT. EOF()
    If Startbal > 0
        DO Invoice WITH Company, Startbal, Baldate
    ENDIF
    SKIP
ENDDO

PROCEDURE Invoice
PARAMETERS Company, Startbal, Baldate
? Company, Startbal, Baldate
RETURN
```

Portability

Not supported in dBASE III PLUS. Nested SCAN loops aren't supported in dBASE IV.

See Also

DO WHILE, DO...UNTIL, FOR...NEXT, INDEX, LOCATE, SEEK, SKIP

SECONDS()

Date and time data

Returns the number of seconds that have elapsed on your computer's system clock since 12 a.m. (midnight).

Syntax

SECONDS()

Description

SECONDS() returns the number of seconds to the hundredth of a second that have elapsed on your system clock since 12a.m.(midnight). The number is in the format SS.hh, where SS are the seconds, and hh are the hundredths of a second.

Use SECONDS() to calculate the amount of time that portions of your program take to run. SECONDS() is more convenient for this purpose than TIME() because SECONDS() returns a number rather than a character string.

You can also use `SECONDS()` instead of `ELAPSED()` to determine elapsed time for the current day to within hundredths of a second.

Example

The first line of the following example parses the hour, minute and second characters of the string returned by `TIME()` to calculate the number of seconds elapsed between midnight and the current time. The second command line is meant to illustrate that `SECONDS()` returns the same value as calculated by the first command line:

```
? TIME(), (VAL(SUBSTR(TIME(),1,2))*3600) + ;
    (VAL(SUBSTR(TIME(),4,2))*60) + ;
    VAL(SUBSTR(TIME(),7,2))
? SECONDS()
```

See Also

`ELAPSED()`, `SET TIME`, `TIME()`

SEEK

Table organization

Searches for the first record in an indexed table whose key fields matches the specified expression.

Syntax

`SEEK <expC list> | <expN list>`

<expC list> | <expN list> The character string or number to find in the master index key fields. For dBASE tables, you can specify a dBASE expression that matches the index key expression. For Paradox and SQL tables, you can specify values (separated by commas) that match single or composite index key fields.

Description

Visual dBASE can search a table for specific information either by a sequential search of a table or by an indexed search of the table's master index. A sequential search is similar to looking for information in a book by reading the first page, then the second, and so on, until the information is found or all pages have been read. `LOCATE` uses this method, checking each record until the information is found or the last record has been inspected.

An indexed search is similar to looking up a topic in a book index and turning directly to the appropriate page. Once a table index is created, `SEEK`, or the similar command `FIND`, can use this index to quickly identify the appropriate record.

`SEEK` begins searching at the top of an index and halts when either a match is found or the end of the index is reached. If a match is found (`FOUND()` returns .T.), the record pointer of the associated table is positioned at the record containing the match.

Use `SKIP` to access other records whose key fields match the index key fields or expression. `SKIP` advances the record pointer one record; because of the indexed order, other matches immediately follow the first. However, `SKIP` after `SEEK` (unlike

S

CONTINUE after LOCATE) doesn't search for a match; it moves the record pointer one record whether or not it finds a match.

The SET NEAR setting determines whether *Visual* dBASE, after an unsuccessful SEEK, positions the record pointer at the end-of-file or at the record in the indexed table immediately after the position at which the value searched for would have been found. If SET NEAR is OFF (the default) and SEEK is unsuccessful, EOF() returns .T. and FOUND() returns .F. If SET NEAR is ON and SEEK is unsuccessful, EOF() returns .F. (unless the position at which the sought value would have been found is the last record in the index), and FOUND() returns .F.

SEEK can locate any valid key fields or expressions of data type character, numeric, float, or date. Character input must be delimited with single or double quotation marks or brackets. Date expressions must be delimited with braces ({ }) or converted using the CTOD() function.

The expression you look for with SEEK must match the key expression or fields of the master index. For example, if the master index key is SUBSTR(Custno, 2, 5), use SEEK SUBSTR(Custno, 2, 5).

When you seek a key expression of type character, the rules established by SET EXACT determine if a match exists. If SET EXACT is OFF, the default condition, only the beginning characters of the key field need to be entered for SEEK to determine a match. If SET EXACT is ON, the entered expression must be identical to the key field for a match to exist.

The SEEK() function works like SEEK followed by FOUND(), except that SEEK searches in the current work area, while SEEK() can search in a current or a specified work area. SEEK() returns .T. or .F. depending on whether the search is successful.

FIND, SEEK, and LOCATE each have their own advantages. FIND and SEEK conduct the most rapid searches; however, both require an indexed file and can search only for values of the key expression. SEEK offers greater flexibility than FIND by accepting dBASE expressions as well as character and numeric input. Searches for memory variable values using FIND require use of the & macro-substitution operator, while character input to SEEK must be delimited.

If the information for which you are searching is in an unindexed file or is not contained in the key field of an indexed file, you can use LOCATE. LOCATE accepts an expression of any data type as input and can search any field of a table for that value. For large tables, however, a sequential search using LOCATE can be slow. In such cases, you might want to use INDEX to create a new index and then use SEEK, SEEK(), or FIND.

Example

The following example uses SEEK to locate the first occurrence of a company in Illinois:

```
USE Company EXCLUSIVE
INDEX ON State_Prov TAG State
Keyvalue="IL"
SEEK Keyvalue      && SEEK the contents of a variable
IF FOUND()
    ? "All Companies in Illinois"
```



```

LIST FIELDS Company, State_Prov;
  WHILE State_Prov=Keyvalue
ELSE
  ? "Illinois was not found"
ENDIF

```

See Also

DTOS(), EOF(), FIND, FOUND(), INDEX, LOCATE, SEEK(), SET EXACT, SET NEAR

SEEK()

Table organization

Searches for the first record in an indexed table whose key field matches the specified expression.

Syntax

SEEK(<expC> | <expN> [<alias>])

<expC> | <expN> The character string or number to look for in the key field of the indexed table in the current or <alias> work area. Character input must be delimited with single or double quotation marks or brackets. Date expressions must be delimited with braces ({ }) or converted using the CTOD() function.

<alias> A work area number (1 through 225), letter (A through J), or alias name. The work area letter or alias name must be enclosed in quotes.

Description

SEEK() evaluates the expression specified by <expC> or <expN> and attempts to find its value in the master index of the table open in the current or specified work area. SEEK() returns .T. if it finds a match of the key expression in the master index, and .F. if no match is found.

SEEK() can search only for an expression that is contained in the master index key field. When you specify <expC>, a character expression, and SET EXACT is ON, <expC> must match every character in the key field. If SET EXACT is OFF, Visual dBASE compares <expC> only to the first characters in the key field.

SEEK() begins searching at the top of an indexed table and moves the record pointer to the first record whose key field matches <expC> or <expN>. If Visual dBASE doesn't find <expC> or <expN>, SEEK() moves the record pointer to the end of the file. SEEK() can locate any valid key expression of data type character, numeric, float, or date. If SET NEAR is ON, the record pointer moves to the record immediately following the expression searched for whenever an exact match cannot be found.

Use SKIP to access other records whose key field value matches the search expression. SKIP advances the record pointer one record. Because of the indexed order, other matches immediately follow the first. However, SKIP after SEEK() (unlike CONTINUE after LOCATE) doesn't search for a match; it moves the record pointer one record whether or not it finds a match.

S

SELECT

You can use SEEK() to produce the same effect as performing SEEK followed by FOUND(). The value that FOUND() returns matches that returned by SEEK().

Example

The following example uses SEEK() to find the first occurrence of a matching record in an indexed table:

```
CLOSE DATABASE
USE Company EXCLUSIVE      && work area 1
INDEX ON State_Prov TAG State
SELECT 2
Gotit=SEEK("CA", "Company")
```

dBASE is pointing to work area 2 and performing a SEEK() in work area 1.

Portability

Not supported in dBASE III PLUS.

See Also

EOF(), FIND, FOUND(), INDEX, LOOKUP(), SEEK, SET EXACT, SET NEAR, SET RELATION, SET SKIP

SELECT

Table basics

Sets the current work area in which to open or perform operations on a table.

Syntax

SELECT <alias>

<alias> Specifies a work area. You can enter a work area number (1 through 225), letter (A through J), or alias name. The work area letter or alias name can also be enclosed in quotes.

Description

Use SELECT to choose a work area in which to open a table, or to specify a work area in which a table is already open. Use SELECT to open a table and other files associated with a table, such as index, query, and format files, in each work area.

Each work area supports its own value of FOUND() and an independent record pointer. Changes in the record pointer of the active work area have no effect on the record pointers of any other work areas, unless you set a relation between the work areas with the SET RELATION command.

Use the macro (&) operator to select a work area by a variable, or simply enclose the variable containing the work area in parentheses. For example, if N=1, use SELECT (N) to select work area 1.

Example

The following example uses the SELECT command to change the current table to a different previously opened table:

```
PUBLIC table1, table2
USE Contact EXCLUSIVE IN SELECT()
INDEX ON CompCode TAG CompCode
table1 = ALIAS()

USE Company IN SELECT()
table2 = ALIAS()

SELECT Company
COPY STRUCTURE TO CntctLst;
  FIELDS Contact->CompCode, ;
  Company->Company, Contact->Contact, ;
  Company->Street1, Company->Street2, ;
  Company->City, Company->State_Prov,;
  Company->Zip_P_Code
USE CntctLst IN SELECT()
table3 = ALIAS()
APPEND
SELECT &table1
? ALIAS(),DBF(),WORKAREA()
SELECT &table3
? ALIAS(),DBF(),WORKAREA()
SELECT &table2
? ALIAS(),DBF(),WORKAREA()
SELECT 3
? ALIAS(),DBF(),WORKAREA()
SELECT 1
? ALIAS(),DBF(),WORKAREA()
SELECT 2
? ALIAS(),DBF(),WORKAREA()
CLOSE ALL
```

See Also

SELECT(), SET RELATION, USE, WORKAREA()

SELECT()

Table basics

Returns the number of an available work area or the work area number associated with a specified alias.

S**Syntax**

SELECT([<alias>])

<alias> A work area number (1 through 225), letter (A through J), or alias name. The work area letter or alias name must be enclosed in quotes.

Description

If you do not specify an alias, SELECT() returns the number of the next available work area, a number between 1 and 225. If you specify an alias, SELECT() determines whether the specified alias name is already in use.

Use SELECT() to locate an available work area in which to open a table without closing an open table. SELECT() returns a value of 0 when there are no more work areas available or a specified <alias> is not already in use.

Example

The following example demonstrates how SELECT() is used to return the next available work area or the work area of an alias:

```
CLOSE DATABASES
USE Clients IN SELECT()
USE Company IN SELECT()
? SELECT()           && Returns 3, meaning work area 3 is next available
? ALIAS(1)           && Returns CLIENTS
? ALIAS(2)           && Returns COMPANY
? DBF(1)             && Returns C:CLIENTS.DBF
? SELECT()           && Returns 3, Work Area 3 still current work area
SELECT 2             && Work area 2 active
? ALIAS()            && Returns COMPANY
? WORKAREA()         && Returns 2
? SELECT()           && Returns 3
? SELECT("Clients")  && Returns 1 for work area 1
```

Portability

Not supported in dBASE III PLUS.

See Also

ALIAS(), DBF(), SELECT, WORKAREA()

SET

Environment

Displays a dialog box for viewing and changing the values of many SET commands. The changed values are stored in the DBASEWIN.INI file.

Syntax

SET

Description

Use SET to view and change settings interactively, instead of typing individual SET commands such as SET TALK ON in the Command window.

Note Any changes you make to settings by using SET are automatically saved to DBASEWIN.INI. This means that the settings will be in effect each time you start dBASE. If you want to change the value of SET commands only temporarily, issue

individual SET commands in the Command window or in a program. For more information about DBASEWIN.INI, see Appendix C in the *User's Guide*.

Issuing SET is the same as choosing the Properties | Desktop menu option.

Portability

In dBASE IV and dBASE III PLUS, changes you make by using SET are valid only until you exit dBASE; they are not saved in the CONFIG.DB file.

See Also

DISPLAY STATUS, SET(), SETTO(), individual SET commands

SET ALTERNATE

Input/Output

Controls the recording of input and output in an alternate text file.

Syntax

SET ALTERNATE on | OFF

SET ALTERNATE TO

[<filename> | ? | <filename skeleton>

[ADDITIVE]]

<filename> | ? | <filename skeleton> The alternate text file, or target file, to create or open. The ? and <filename skeleton> options display a dialog box in which you can specify a new file or select an existing file. If you specify a file without including its path, dBASE looks for the file in the current directory, then in the path you specify with SET PATH. If you specify a file without including its extension, dBASE assumes .TXT.

ADDITIVE Appends dBASE output that appears in the results pane of the Command window to the specified existing alternate file. If the file doesn't exist, dBASE returns an error message.

Default

The default for SET ALTERNATE is OFF. To change the default, set the ALTERNATE parameter in the [OnOffSetting Settings] section of DBASEWIN.INI. To set a default file name for use with SET ALTERNATE, specify an ALTERNATE parameter in the [CommandSettings] section of DBASEWIN.INI.

Description

Use SET ALTERNATE TO to create a record of dBASE output and commands. You can edit the contents of this file with the Text Editor for use in documents, or store it on disk for future reference. You can record, edit, and incorporate command sequences into new programs. (To send the results of @...SAY commands to a text file, use SET DEVICE TO FILE.)

SET ALTERNATE TO <filename> only opens an alternate file, while SET ALTERNATE ON | OFF controls the storage of input and output to that file. Only one alternate file

can be open at a time. When you issue SET ALTERNATE TO *<filename>* to open a new file, dBASE closes the previously open alternate file.

When SET ALTERNATE is ON, dBASE stores output to the results pane of the Command window in the text file you've opened by previously issuing SET ALTERNATE TO *<filename>*. An alternate file must be open for SET ALTERNATE ON to have an effect. SET ALTERNATE doesn't affect a program's output; it only determines when that output is saved in the alternate file. (Keyboard entries in the Command window aren't stored to the alternate file.)

To prevent your text file from beginning with a blank line, use two question marks (??) before the first word that you send to the alternate file.

Issuing SET ALTERNATE OFF does not close the alternate file. Before accessing the contents of an alternate file, formally close it with CLOSE ALTERNATE or SET ALTERNATE TO (with no file name). This ensures that all data recorded by dBASE for storage in the alternate file is transferred to disk, and automatically turns SET ALTERNATE to OFF.

If SET SAFETY is ON, you don't use the ADDITIVE option, and a file exists with the same name as the target file, dBASE displays a dialog box asking if you want to overwrite the file. If SET SAFETY is OFF and you don't use the ADDITIVE option, any existing file with the same name as the target file is overwritten without warning.

Example

This example uses the SET ALTERNATE commands to write text to the screen and an ASCII file:

```
SET ALTERNATE TO Rose
* Open Rose.txt for text output
? "Opening alternate file" && to screen only
SET ALTERNATE ON
* ?,?? commands now go to Rose.txt
? "A rose "
SET ALTERNATE OFF && Stop storing to Rose.txt.
?? "tended carefully in your garden "
SET ALTERNATE ON && Add to Rose.txt.
?? "is a rose "
?? "is a rose "
? "You will be proud of"
CLOSE ALTERNATE && Close Rose.txt
* Rose.txt contains:
* A rose is a rose is a rose
* You will be proud of
```

Portability

The ADDITIVE option is not supported in dBASE III PLUS. The ? and *<filename skeleton>* options are not supported in dBASE III PLUS or dBASE IV.

See Also

CLOSE..., SET DEVICE

SET AUTOSAVE

Fields and records

Determines if dBASE writes data to disk each time a record is changed or added.

Syntax

SET AUTOSAVE on | OFF

Default

The default for SET AUTOSAVE is OFF. To change the default, update the AUTOSAVE setting in DBASEWIN.INI. To do so, either use the SET command to specify the setting interactively, or enter the AUTOSAVE parameter directly in DBASEWIN.INI.

Description

Use SET AUTOSAVE ON to reduce the chances of data loss. When SET AUTOSAVE is ON and you alter or add a record, *Visual* dBASE updates tables and index files on disk when you move the record pointer. When SET AUTOSAVE is OFF, changes are saved to disk as the record buffer is filled.

Since *Visual* dBASE periodically saves table changes to disk, in most situations you don't need to SET AUTOSAVE ON. SET AUTOSAVE OFF lets you process data faster, since *Visual* dBASE writes your changes to disk less often.

Example

The following example uses SET AUTOSAVE ON to save newly entered data to disk each time the record pointer changes:

```
SET TALK OFF
CLOSE DATABASES
USE Company
a_save = SET("AUTOSAVE")
carry = SET("CARRY")
SET AUTOSAVE ON
SET CARRY ON
APPEND
SET CARRY &carry
SET AUTOSAVE &a_save
FLUSH
CLOSE ALL
CLEAR
SET TALK ON
```



Portability

Not supported in dBASE III PLUS.

See Also

CLOSE..., FLUSH

Turns the computer bell on or off and sets the bell frequency and duration.

Syntax

SET BELL ON | off

SET BELL TO

[<frequency expN>, <duration expN>]

<frequency expN> The frequency of the bell tone in cycles per second, which must be an integer from 19 to 10,000, inclusive.

<duration expN> The duration of the bell tone in ticks (18ths of a second), which must be an integer from 1 to 19, inclusive.

Default

The default for SET BELL is ON. The default bell frequency is 512 Hertz (cycles per second), and the default bell duration is 2 ticks. (A tick is 1/18 of a second.)

To change the default on | off setting, update the BELL parameter in the [OnOffCommandSettings] section of DBASEWIN.INI. To change the frequency and duration defaults, update the BELL parameters in the [CommandSettings] section of DBASEWIN.INI. To do so, either use the SET command to specify the setting interactively, or enter the parameters directly in DBASEWIN.INI.

Description

When SET BELL is ON, dBASE produces a tone when you fill a data entry field or enter invalid data. SET BELL TO determines the frequency and duration of this tone.

CHR(7) sounds a tone whether SET BELL is ON or OFF.

SET BELL TO with no arguments sets the frequency and duration to the default values of 512 and 2. SET BELL has an effect only on machines that have an internal speaker or other sound system.

Example

The following examples set the bell to high and low pitch and short and long durations:

```
SET BELL ON
SET BELL TO 50,19
* BROWSE, APPEND etc will now ring the bell
* A long, very low bell
? CHR(7)  && ring the bell
SET BELL TO 10000,1
* Short, very high pitched
? CHR(7)  && ring the bell
SET BELL OFF
? CHR(7)  && the bell still rings
* BROWSE, APPEND etc will not ring the bell
```


Portability

SET BELL TO *<frequency expN>*, *<duration expN>* not supported in dBASE III PLUS.

See Also

CHR(), SET CONFIRM

SET BLOCKSIZE

Fields and records

Changes the default block size of memo field and .MDX index files.

Syntax

SET BLOCKSIZE TO *<expN>*

<expN> A number from 1 to 63 that sets the size of blocks used in memo and .MDX index files. (The actual size in bytes is the number you specify multiplied by 512.)

Default

The default for SET BLOCKSIZE is 1 (for compatibility with dBASE III PLUS). To change the default, update the BLOCKSIZE setting in DBASEWIN.INI.

Description

Use SET BLOCKSIZE to change the size of blocks in which *Visual* dBASE stores memo field files and .MDX index files on disk. The actual number of bytes used in blocks is *<expN>* multiplied by 512. Instead of using SET BLOCKSIZE, you can set the block size used for memo and .MDX index files individually, by using SET MBLOCK and SET IBLOCK commands.

After the block size is changed, memo fields created with the COPY, CREATE, and MODIFY STRUCTURE commands have the new block size. To change the block size of an existing memo field file, use the SET BLOCKSIZE command to change the block size and then copy the table containing the associated memo field to a new file. The new file then has the new block size.

Example

The following example uses SET BLOCKSIZE to create another table that is a copy of Clients but has a memo blocksize of 1024 bytes embedded in its structure instead of the default of 512 bytes:

```
USE Clients
? SET("Blocksize")           && Returns 1, each memo block = 512 bytes
SET BLOCKSIZE TO 2
COPY TO Clients2
USE Clients2
? SET("Blocksize")           && Returns 2
LIST FILES LIKE *.DBT         && Note file size larger than Clients.DBT
CLOSE DATABASES
```

S

Portability

Not supported in dBASE III PLUS.

See Also

COPY, COPY INDEXES, CREATE, MODIFY STRUCTURE, INDEX, REINDEX, REPLACE, SET(), SET IBLOCK, SET MBLOCK

SET BORDER

Environment

Defines the default border of subsequently defined dBASE IV windows, dBASE IV pop-up menus, and @...TO boxes. This command is supported primarily for compatibility with dBASE IV. In dBASE for Window, use DEFINE to create *forms* instead of windows.

For more information about SET BORDER, see online Help. For more information about working with forms, see the Forms chapters in the *User's Guide*.

SET CARRY

Fields and records

Specifies the fields from which data is copied to new records created with APPEND, BROWSE, EDIT, or INSERT commands.

Syntax

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

SET CARRY on | OFF

<field list> The list of fields whose values are carried forward to new records.

ADDITIVE Adds fields in *<field list>* to the list of fields previously defined with SET CARRY TO. Without ADDITIVE, *<field list>* overrides any previously specified list of fields.

Default

The default for SET CARRY is OFF.

Description

When SET CARRY is ON, records added using APPEND, BROWSE, EDIT, or INSERT are filled with the contents of the record immediately preceding the new record at the logical end of the table. (You can also supply default record contents for INSERT and APPEND using the DEFAULT argument of @...SAY...GET.)

When SET CARRY is OFF, new records are initially blank. SET CARRY doesn't affect INSERT AUTOMEM, APPEND AUTOMEM, INSERT BLANK, or APPEND BLANK. These commands always add a record containing automem variables or a blank record.

Specifying a field list with the SET CARRY TO command limits the fields carried to the new record. Using the SET CARRY TO command automatically sets CARRY ON.

Specifying the ADDITIVE keyword adds fields to an already defined field list.

SET CARRY TO with no field list restores the default condition where all fields are carried to new records.

You can also limit the fields by specifying a field list with SET FIELDS TO. The SET CARRY TO field list applies only to SET CARRY, while the SET FIELDS TO fields list applies to all commands operating on tables. A SET FIELDS TO field list overrides a SET CARRY TO field list.

Example

The following example uses SET CARRY TO to carry field data forward from the previous record. In the example, the user wants to enter new records for companies that are all in the same local area so they have the same State and Zip, plus report to the same corporate office:

```
USE Company
SET CARRY TO State_Prov, Zip_P_Code, Corp_Off
APPEND
* Edit window appears with data already entered in the 3 designated fields.
SET CARRY OFF
CLOSE ALL
```

Portability

Not supported in dBASE III PLUS.

See Also

APPEND, INSERT, SET FIELDS, @...SAY...GET

SET CATALOG

Table basics

Opens a catalog file.

Syntax

SET CATALOG TO [*<filename>* | ?]

SET CATALOG on | OFF

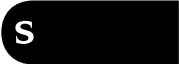
TO *<filename>* | ? Specifies the name of the catalog you want to open. SET CATALOG TO ? displays a dialog box, in which you can select an existing catalog file.

Default

The default for SET CATALOG is OFF. To change the default, update the CATALOG setting in DBASEWIN.INI. To do so, either use the SET command to specify the setting interactively, or enter the CATALOG parameter directly in DBASEWIN.INI.

Description

Use SET CATALOG TO *<filename>* to open an existing catalog.



SET CATALOG ON|OFF determines if new files are added to the open catalog file. After setting CATALOG ON, all tables and associated files such as index, query, format, report, and label files you use or new files you create are added to the catalog.

To stop adding files to an open catalog, use SET CATALOG OFF. Then, when you want to resume adding files to the catalog, use SET CATALOG ON again. To close a catalog, use the SET CATALOG TO command without a filename.

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

When you select a catalog file, it is automatically opened in its own work area buffer. If you want to manipulate the catalog table, first open it in a user-accessible work area:

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

To update the catalog from within a program, temporarily disable the catalog:

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

Whenever you open a catalog, dBASE checks the catalog contents against the disk. If you've previously deleted any files with SET CATALOG OFF, dBASE deletes the corresponding entries in the catalog for the files that have been deleted.

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	INDEX
COPY STRUCTURE EXTENDED	JOIN
CREATE	SET FILTER
CREATE FROM	SET FORMAT
CREATE MODIFY FORM	SET VIEW
CREATE MODIFY LABEL	SORT
CREATE MODIFY REPORT	TOTAL
CREATE MODIFY VIEW	USE
IMPORT FROM	

If the file name already exists in the catalog, you are prompted for a file title (if SET TITLE is ON). Use the SET TITLE OFF command to suppress the file title prompt if you do not want to enter file titles.

Example

The following example first CREATES a CATALOG that contains only the Company and Orders tables. It later opens the catalog with SET CATALOG TO so that a user does not need to deal with any other tables. They can use the Catalog viewer in place of the file viewer:

```
CLOSE ALL
CREATE CATALOG CompOrd
USE Company
USE Orders IN Select()
SET CATALOG TO
* Now CompOrd is created

SET CATALOG TO CompOrd
```

Now user can use Catalog Viewer and need not deal with other tables in the subdirectory.

See Also

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

SET CENTURY

Date and time data

Controls the format in which dBASE displays the year portion of dates.

Syntax

SET CENTURY on | off

Default

The default for SET CENTURY is set by the International option of the Windows Control Panel. To change the default, set the CENTURY parameter in DBASEWIN.INI. To do so, either use the SET command to specify the setting interactively, or enter the CENTURY parameter directly in DBASEWIN.INI.

Description

When SET CENTURY is ON, dBASE displays dates in the current format with 4-digit years; when SET CENTURY is OFF, dBASE displays dates in the current format with 2-digit years.

You can enter a date with a 2-, 3-, or 4-digit year whether SET CENTURY is ON or OFF. dBASE assumes that 2-digit years are in the 20th century. If SET CENTURY is OFF, dBASE truncates any digits to the left of the last two when displaying the date. However, dBASE stores the correct value of the date internally.

S

SET COLOR OF

The following table shows the how dBASE displays and stores dates depending on the setting of SET CENTURY. (The table assumes SET DATE is AMERICAN.)

You enter date as	dBASE stores date as	With SET CENTURY ON, dBASE displays	With SET CENTURY OFF, dBASE displays
{10/13/94}	10/13/1994	10/13/1994	10/13/94
{10/13/994}	10/13/0994	10/13/0994	10/13/94
{10/13/1994}	10/13/1994	10/13/1994	10/13/94
{10/13/2094}	10/13/2094	10/13/2094	10/13/94

As the table shows, SET CENTURY doesn't affect the relationship between how you enter a date and how dBASE evaluates and stores it. SET CENTURY affects only how dBASE *displays* the year portion of the date.

Example

See the example of SET DATE for an example of SET CENTURY.

See Also

DATE(), SET DATE

SET COLOR OF

Colors and fonts

Defines the color settings for specified screen areas. This command is supported primarily for compatibility with dBASE IV. In *Visual* dBASE, use the ColorNormal or ColorHighlight properties to define colors for objects.

For more information about SET COLOR OF, see online Help. For more information about working with *Visual* dBASE objects, see Chapter 10 in the *Programmer's Guide*.

SET COLOR TO

Colors and fonts

Defines color display characteristics for dBASE IV windows and for text that appears in the results pane of the Command window. This command is supported primarily for compatibility with dBASE IV. In *Visual* dBASE, use the ColorNormal or ColorHighlight properties to define colors for objects.

For more information about SET COLOR TO, see online Help. For more information about working with *Visual* dBASE objects, see Chapter 10 in the *Programmer's Guide*.

SET CONFIRM

Environment

Controls the cursor's movement from one entry field to the next during data entry in the results pane of the Command window and in dBASE IV windows. This command is

supported primarily for compatibility with dBASE IV. It has no effect on *Visual* dBASE forms.

For complete syntax information on SET CONFIRM, see online Help. For more information about working with forms, see the Forms chapters in the *User's Guide*.

SET CONSOLE

Environment

Controls the display of input and output in the results pane of the Command window during program execution.

Syntax

SET CONSOLE ON | off

Default

The default for SET CONSOLE is ON.

Description

When SET CONSOLE is ON, dBASE displays all standard input and command output in the results pane of the Command window. Use SET CONSOLE OFF to prevent dBASE from displaying this input and output.

You can issue SET CONSOLE only in a program, not in the input pane of the Command window. SET CONSOLE does not affect the display of error messages or safety prompts, nor does it affect the display of input entered in the input pane of the Command window.

A user can enter input requested by a program (such as by WAIT or ACCEPT) while SET CONSOLE is OFF; however, dBASE displays neither the prompt for the input nor the input itself.

@...SAY...GETs override the SET CONSOLE setting and are visible regardless of the SET CONSOLE status.

Example

This example uses SET CONSOLE OFF to turn off the screen display while a report is sent to the printer. It then sets the console back on:

```
SET CONSOLE OFF
REPORT FORM Report1 TO PRINTER
SET CONSOLE ON
```

Turn CONSOLE OFF only for a specific reason.

See Also

ACCEPT, ON ERROR, SET TALK, WAIT

SET COVERAGE

Determines whether dBASE creates/updates a coverage file (.COV).

Syntax

SET COVERAGE on | OFF

Default

The default for SET COVERAGE is OFF. To change the default, set the COVERAGE parameter in DBASEWIN.INI. To do so, either use the SET command to specify the setting interactively, or enter the COVERAGE parameter directly in DBASEWIN.INI.

Description

A coverage file is a binary file containing cumulative information on how many times, if any, dBASE enters and exits (and thus fully executes) each logical block of a program. Use SET COVERAGE as a program development tool to determine which program lines dBASE executes and doesn't execute each time you run a program.

When SET COVERAGE is ON, and you call a program, procedure, or user-defined function (UDF) in a separate program file, dBASE creates a new coverage file or updates an existing one. When dBASE creates a coverage file, it assigns the file the root name of the program or procedure file and a .COV extension.

To view the contents of a coverage file, use DISPLAY COVERAGE or LIST COVERAGE. If the coverage file reveals that some lines aren't executing, you can respond by changing the program or the input to the program to make the lines execute. In this way, you can make sure that you test all lines of code in the program.

You can issue SET COVERAGE ON in a program file or in the Command window. If you issue SET COVERAGE ON in a program file, dBASE will create coverage files for that program and for each subroutine that exists in separate programs or procedure files that the main program calls. You can also issue #pragma COVERAGE(ON) in a program file to create a coverage file for it and all program files it calls.

If you issue SET COVERAGE ON in the Command window, dBASE will create coverage files for any programs, or procedures or user-defined functions (UDFs) in open procedure files, that you call from the Command window until you issue SET COVERAGE OFF.

A logical block doesn't include commented lines or programming construct command lines such as IF and ENDIF. It does, however, include command lines within programming construct command lines. If your program doesn't contain any programming constructs (IF, DO WHILE, FOR...NEXT, SCAN...ENDSCAN, LOOP, DO CASE, DO...UNTIL), the program has only one logical block consisting of all uncommented command lines.

The coverage file identifies a logical block by its corresponding program line number(s):

```
Line 1  * UPDATES.PRG
Line 2  SET TALK OFF Block 1 (Lines 2-3)
Line 3  USE Customer INDEX Salespers
```



```

Line 4  SCAN
Line 5      DO CASE
Line 6      CASE Salesper = "S-12"
Line 7          SELECT 2 Block 2 (Lines 7-8)
Line 8          USE S12
Line 9      CASE Salesper = "L-5"
Line 10         SELECT 2 Block 3 (Lines 10-11)
Line 11         USE L5
Line 12      CASE Salesper = "J-25"
Line 13         SELECT 2 Block 4 (Lines 13-14)
Line 14         USE J25
Line 15      ENDCASE
Line 16      DO Changes Block 5 (Lines 16-17)
Line 17      SELECT 1
Line 18  ENDSCAN
Line 19  CLOSE ALL  Block 6 (Lines 19-20)
Line 20  SET TALK ON

```

Before dBASE can create coverage files, you must compile the program file and all called files you want to analyze while SET COVERAGE is ON. If you COMPILE a program while SET COVERAGE is OFF, then turn SET COVERAGE ON and DO the program, dBASE doesn't produce a coverage file.

dBASE writes the coverage file to disk when the program is unloaded from memory or when you issue a LIST COVERAGE or DISPLAY COVERAGE. To unload a program from memory, use CLEAR PROGRAM.

The preprocessor command #pragma COVERAGE(ON) can only be used in a program file. When you use #pragma COVERAGE(ON), you don't have to issue SET COVERAGE ON before compiling the program; instead of recreating the coverage file, dBASE updates the existing one when you make changes to the program and recompile.

Example

The following example is a batch program that updates records from a central file:

```

* Update.PRG
USE Clients
SET COVERAGE ON
SCAN
  IF DELETED()
    DO Deletes
  ELSE
    DO Updates
  ENDIF
ENDSCAN
DISPLAY COVERAGE Update
CLOSE ALL

PROCEDURE Updates
  * update code here

PROCEDURE Deletes
  * delete code here

```

DISPLAY COVERAGE opens a .COV file that shows which lines of the program dBASE executed when the program was run. If no records were marked for deletion, the coverage file would reveal that dBASE had not executed one logical block, the line DO DELETES. To test whether the call to procedure Deletes works, the programmer could delete a test record and run the program again so that dBASE executes line 5.

When SET COVERAGE is ON and dBASE executes every line of a program, procedure, or UDF file, the coverage file states that test coverage was 100%.

Portability

Not supported in dBASE IV or dBASE III PLUS.

See Also

#pragma, CLEAR PROGRAM, COMPILE, DEBUG, DISPLAY COVERAGE, SET DEVELOPMENT

SET CUAENTER

Forms

Determines whether *Enter* works in Windows mode or dBASE DOS mode.

Syntax

SET CUAENTER ON | off

Default

The default for SET CUAENTER is ON. To change the default, update the CUAENTER setting in DBASEWIN.INI. To do so, either use the SET command to specify the setting interactively, or enter the CUAENTER parameter directly in DBASEWIN.INI.

Description

Use SET CUAENTER to control *Enter*.

In *Visual* dBASE, pressing *Enter* submits the current form; this is typical of Windows applications. However, in dBASE DOS, pressing *Enter* moves the cursor from one GET field to the next. In *Visual* dBASE, you move focus from object to object with the mouse or by pressing *Tab* and *Shift-Tab* instead of *Enter*.

To move focus in the dBASE DOS manner, execute SET CUAENTER OFF. You typically do this when you need to maintain consistency with your DOS applications.

Example

The following example creates a basic form with three entry fields from the Contact table and two pushbuttons to advance or retard the record pointer. SET CUAENTER OFF is used to make the cursor behave as it would in a DOS application, that is, advance between fields or to the pushbuttons when the user presses Enter. Changing the command to SET CUAENTER ON causes the cursor to respond only to the TAB key and mouse clicks:

```

SET PROCEDURE TO PROGRAM(1) ADDITIVE
DEFINE FORM Main FROM 0,0 to 13,40;
    PROPERTY ColorNormal "BG+/BG",;
    Text "System Update"
DEFINE TEXT T1 OF Main AT 3,5 ;
    PROPERTY TEXT "Enter Current Time (24 hour):",;
    Width 30
DEFINE ENTRYFIELD F1 OF Main AT 3,28 ;
    PROPERTY Value SPACE(8), Picture "99:99:99",;
    Width 8
DEFINE TEXT T2 OF Main AT 5,5 ;
    PROPERTY TEXT "Enter Current Date",;
    Width 20
DEFINE ENTRYFIELD F2 OF Main AT 5,28 ;
    PROPERTY Value {}, Picture "99/99/99",;
    Width 8
DEFINE PUSHBUTTON Update OF Main AT 9,7;
    PROPERTY TEXT "Update System Time and Date",;
    Height 2, Width 26, OnClick Update
OPEN FORM Main

PROCEDURE Update
SET DATE TO DTOC(Form.F2.Value)
SET TIME TO Form.F1.Value
? "Update complete"
CLOSE FORM Main
RETURN

```

Portability

Not supported in dBASE IV or dBASE III PLUS.

See Also

READ, @...SAY...GET

SET CURRENCY

Numeric data

SET CURRENCY left | right positions currency symbol(s) to the left or right of a monetary value when you use the "\$" PICTURE or FUNCTION symbol or the function TRANSFORM(). SET CURRENCY TO determines the characters dBASE uses for the currency symbol.

Syntax

SET CURRENCY left | right

SET CURRENCY TO
[<expC>]

LEFT Places currency symbol(s) to the left of currency numbers.

RIGHT Places currency symbol(s) to the right of currency numbers.

S

<expC> The characters that appear as a currency symbol. Although dBASE imposes no limit to the length of *<expC>*, it recognizes only the first nine characters. You can't include numbers in *<expC>*.

Default

The defaults for SET CURRENCY are set by the International option of the Windows Control Panel. You can change the defaults interactively by using the SET command, or by specifying parameters directly in DBASEWIN.INI.

- To change the default location of the currency symbol, set the CURRENCY parameter to LEFT or RIGHT in the [OnOffCommandSettings] section in DBASEWIN.INI.
- To change the default characters that appear as a currency symbol, set the CURRENCY parameter in the [CommandSettings] section in DBASEWIN.INI.

Description

Use SET CURRENCY left | right to specify the position of currency symbol(s) in monetary numeric values. Use SET CURRENCY TO to establish a currency symbol other than the default.

When SET CURRENCY is LEFT, dBASE displays only as many currency symbols as fit, together with the digits to the left of any decimal point, within ten character spaces.

SET CURRENCY TO without the *<expC>* option resets the currency symbol to the default set with the International option of the Windows Control Panel.

Example

An example of SET CURRENCY LEFT | RIGHT is shown in the example for INT().

Portability

SET CURRENCY left | right is not supported in dBASE III PLUS. SET CURRENCY TO *<expC>* is not supported in dBASE III PLUS or dBASE IV.

See Also

SET POINT, SET SEPARATOR, TRANSFORM()

SET CURSOR

Keyboard and mouse events

Determines whether the cursor is visible or hidden.

Syntax

SET CURSOR ON | off

Default

The default for SET CURSOR is ON. To change the default, set the CURSOR parameter in DBASEWIN.INI.

Description

Use SET CURSOR to control whether a cursor appears on the screen. When SET CURSOR is OFF, the cursor is hidden. Issue SET CURSOR ON to redisplay the cursor.

Example

The following program ensures that the cursor is on during the program and resets the cursor when leaving:

```
OldCursor=SET("CURSOR")    && save current setting
SET CURSOR ON              && set cursor on
* ...
SET CURSOR &OldCursor      && reset cursor
```

Portability

Not supported in dBASE III PLUS.

SET DATABASE

Table basics

Sets the default database from which tables are accessed.

Syntax

SET DATABASE TO [*<database name>*]

<database name> Specifies the name of the database you want to make the current database.

Description

SET DATABASE sets the current database, which defines the default location for tables accessed by dBASE commands. Using this command, you can select from any databases previously opened with the OPEN DATABASE command. Databases are defined using the BDE Configuration Utility. (For more information on using this program, see *Getting Started*.)

When you issue the SET DATABASE TO command without a database, *Visual* dBASE restores operation to accessing tables in the current directory (or in the directory specified by SET PATH).

Example

See OPEN DATABASE and DATABASE() for examples of using SET DATABASE.

Portability

Not supported in dBASE IV or dBASE III PLUS.

See Also

CLOSE..., DATABASE(), OPEN DATABASE, SET DBTYPE

SET DATE

Specifies the format dBASE uses for the display and entry of dates.

Syntax

SET DATE [TO]
AMERICAN | ANSI | BRITISH | FRENCH | GERMAN | ITALIAN | JAPAN | USA | MDY | DMY | YMD

TO Include for readability only; TO has no affect on the operation of the command.

AMERICAN | ANSI | BRITISH | FRENCH | GERMAN | ITALIAN | JAPAN | USA | MDY | DMY | YMD The options correspond to the following formats:

Option	Format
AMERICAN	MM/DD/YY
ANSI	YY.MM.DD
BRITISH	DD/MM/YY
FRENCH	DD/MM/YY
GERMAN	DD.MM.YY
ITALIAN	DD-MM-YY
JAPAN	YY/MM/DD
USA	MM-DD-YY
MDY	MM/DD/YY
DMY	DD/MM/YY
YMD	YY/MM/DD

Default

The default for SET DATE is set by the International option of the Windows Control Panel. To change the default, set the DATE parameter in DBASEWIN.INI. To do so, either use the SET command to specify the setting interactively, or enter the DATE parameter directly in DBASEWIN.INI.

Description

SET DATE determines how dBASE displays fields and memory variables of date type. If SET CENTURY is ON, dBASE displays all formats with a 4-digit year.

SET DATE overrides any prior SET MARK setting. However, you can use SET MARK after SET DATE to change the date separator character.

Example

The following examples use SET DATE and SET CENTURY to control the display of date data.

```
date = {04/01/94}           && or date = {4/1/94}
SET CENTURY OFF
SET DATE AMERICAN
? date                      && Returns 04/01/94
SET CENTURY ON
```

? date	&& Returns 04/01/1994
SET DATE JAPAN	
? date	&& Returns 1994/04/01
SET DATE GERMAN	
? date	&& Returns 01.04.1994
SET DATE BRITISH	
? date	&& Returns 01/04/1994
SET DATE ITALIAN	
? date	&& Returns 01-04-1994
SET DATE FRENCH	
? date	&& Returns 01/04/1994
SET DATE USA	
? date	&& Returns 04-01-1994

Portability

The format options JAPAN, USA, MDY, DMY, and YMD aren't supported in dBASE III PLUS.

See Also

DATE(), DMY(), MDY(), SET CENTURY, SET DATE TO, SET MARK

SET DATE TO

Date and time data

Sets the system date.

Syntax

SET DATE TO <expC>

<expC> The character expression, in the current date format, to set as the current system date.

Default

The default for the value of the system date is set by the Date/Time option of the Windows Control Panel.

Description

Use SET DATE TO to reset the date on your system clock. Subsequent values of DATE() and the date stamp of files you save reflect the new date.

Enter <expC> in the current date format as determined by SET DATE, DBASEWIN.INI, or the International option of the Windows Control Panel (in that order). That is, SET DATE settings override those in DBASEWIN.INI, and settings in DBASEWIN.INI override those in the Windows Control Panel. Be sure <expC> matches the date format in use when your program runs.

The date must be in the range from January 1, 1980, to December 31, 2099. Because dBASE assumes 2-digit years refer to twentieth-century dates, you must type all 4 digits for the year of dates beyond the twentieth century.

S

Example

The following example uses SET DATE TO and SET TIME TO to update system time after getting input from the user:

```

SET PROCEDURE TO PROGRAM(1) ADDITIVE
DEFINE FORM Main FROM 0,0 to 13,40;
    PROPERTY ColorNormal "BG+/BG",;
    Text "System Update"
DEFINE TEXT T1 OF Main AT 3,5 ;
    PROPERTY TEXT "Enter Current Time (24 hour):", Width 30
DEFINE ENTRYFIELD F1 OF Main AT 3,28 ;
    PROPERTY Value SPACE(8), Picture "99:99:99", Width 8
DEFINE TEXT T2 OF Main AT 5,5 ;
    PROPERTY TEXT "Enter Current Date", Width 20
DEFINE ENTRYFIELD F2 OF Main AT 5,28 ;
    PROPERTY Value {}, Picture "99/99/99", Width 8
DEFINE PUSHBUTTON Update OF Main AT 9,7;
    PROPERTY TEXT "Update System Time and Date",;
        Height 2, Width 26, OnClick Update
OPEN FORM Main

PROCEDURE Update
SET DATE TO DTOC(Form.F2.Value)
SET TIME TO Form.F1.Value
? "Update complete"
CLOSE FORM Main
RETURN

```

Portability

Not supported in dBASE III PLUS or dBASE IV.

See Also

DATE(), SET CENTURY, SET DATE, SET TIME

SET DBTYPE

Table basics

Sets the default table type to either Paradox or dBASE.

Syntax

SET DBTYPE TO [PARADOX | DBASE]

PARADOX | DBASE Sets the default table type to a Paradox or dBASE table.

Default

The default for SET DBTYPE TO is DBASE. To change the default, update the DBTYPE setting in DBASEWIN.INI. To do so, either use the SET command to specify the setting interactively, or enter the DBTYPE parameter directly in DBASEWIN.INI.

Description

SET DBTYPE sets the default type of table used by commands that open or create a table. You can override this selection by specifying a specific extension, that is, .DBF for a dBASE table or .DB for a Paradox table.

SET DBTYPE TO specified without a DBASE or PARADOX argument returns DBTYPE to its default (dBASE).

Example

The following example uses SET DBTYPE to set the environment for a relation between two .DBF files. Selected fields are copied to a Paradox table. SET DBTYPE then sets the table type environmental variable to Paradox, and the newly created table is used and browsed. SET DBTYPE finally sets the environment back to the default of DBASE:

```
CLOSE DATABASES
SET SAFETY OFF
SET DBTYPE TO DBASE
USE Company ORDER Compcode IN SELECT()
USE Contact IN SELECT()
SELECT Contact
SET RELATION TO CompCode INTO Company
SELECT Contact
COPY TO CntctLst TYPE PARADOX;
    FIELDS Company->Company, Contact->CompCode,;
        Contact->Contact, Company->Street1, Company->Street2, ;
        Company->City, Company->State_Prov, Company->Zip_P_Code
CLOSE DATABASES
SET DBTYPE TO PARADOX
USE CntctLst
BROWSE
CLOSE DATABASES
SET DBTYPE TO
```



Portability

Not supported in dBASE IV or dBASE III PLUS.

See Also

CLOSE..., COPY TABLE, CREATE, DATABASE(), DELETE TABLE, MODIFY STRUCTURE, OPEN DATABASE, RENAME TABLE, SET DATABASE, USE

SET DECIMALS

Numeric data

Determines the number of decimal places of numbers to display.

Syntax

SET DECIMALS TO
[<expN>]

<expN> The number of decimals places, from 0 to 18.

Default

The default for SET DECIMALS is 2. To change the default, set the DECIMALS parameter in DBASEWIN.INI. To do so, either use the SET command to specify the setting interactively, or enter the DECIMALS parameter directly in DBASEWIN.INI.

Description

Use SET DECIMALS to specify the number of decimal places of numbers you want dBASE to display. SET DECIMALS affects the *display* of most mathematical calculations, but not the way numbers are stored on disk or maintained internally.

SET DECIMALS TO without <expN> resets the number of decimal places back to the default of 2.

Example

The following example uses SET DECIMALS to control how many decimal places are displayed:

```
SET DECIMALS TO 2
? 2.22 * 3.3333          && Returns 7.40
SET DECIMALS TO 0
? 2.22 * 3.3333          && Returns 7
SET DECIMALS TO 7
? 2.22 * 3.3333          && Returns 7.3999260
```

Additional examples of using SET DECIMALS are included in the examples for INT() and RANDOM().

See Also

INT(), RANDOM(), ROUND(), SET PRECISION, VAL()

SET DEFAULT

Disk and file utilities

Determines the default drive dBASE uses when searching for and storing files. SET DEFAULT is supported primarily for backward compatibility with dBASE III PLUS. In *Visual* dBASE, you can use SET DIRECTORY to specify a default drive and/or directory in a single statement.

For more information about SET DEFAULT, see online Help.

SET DELETED

Fields and records

Controls whether *Visual* dBASE processes records marked for deletion.

Syntax

SET DELETED ON | off

Default

The default for SET DELETED is ON. To change the default, update the DELETED setting in DBASEWIN.INI. To do so, either use the SET command to specify the setting interactively, or enter the DELETED parameter directly in DBASEWIN.INI.

Description

Use SET DELETED to include or exclude records marked for deletion in a table. When SET DELETED is OFF, all records appear in a table. When SET DELETED is ON, *Visual* dBASE excludes records that have been marked for deletion from processing by subsequent commands such as LOCATE and LIST. Records marked for deletion, however, do remain in the table.

GO <expN>, INDEX, REINDEX, RECCOUNT() and any command executed with the RECORD <expN> scope option aren't affected by SET DELETED. If, however, SET DELETED is ON *and* the records are displayed, GO <expN> does not place the record pointer on a record marked for deletion.

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

Example

The following example uses SET DELETED to exclude records that are marked for deletion from being copied to another table:

```
SET SAFETY OFF
USE Contact EXCLUSIVE IN SELECT()
INDEX ON CompCode TAG CompCode
USE Company IN SELECT()
SELECT Company
SET RELATION TO CompCode INTO Contact
DELETE FOR Company->State_Prov = "CA"
SET DELETED ON
```

S

SET DELIMITERS

```
COPY FIELDS Company->Company, Contact->Contact, ;
Company->Street1, Company->Street2, ;
Company->City, Company->State_Prov,;
Company->Zip_P_Code TO CntctLst TYPE PARADOX
SET DBTYPE TO PARADOX
USE CntctLst
BROWSE
CLOSE ALL
SET DBTYPE TO
SET SAFETY ON
```

Portability

In dBASE IV, the default for SET DELETED is OFF. In addition, regardless of the SET DELETED setting, if you position the record pointer on a specific record with the GO command, dBASE IV displays the record whether or not it is marked for deletion. *Visual* dBASE does not display records marked for deletion when SET DELETED is ON.

See Also

DELETE, DELETED(), PACK, RECALL, SET(), SET FILTER, SET RELATION

SET DELIMITERS

Input/Output

Controls whether specified delimiter characters mark the beginning and end of data-entry fields. This command is supported primarily for compatibility with dBASE IV, and has no effect in *Visual* dBASE forms.

For complete syntax information on SET DELIMITERS, see online Help.

SET DESIGN

Environment

Determines whether CREATE and MODIFY commands can be executed.

Syntax

SET DESIGN ON | off

Default

The default for SET DESIGN is ON. To change the default, set the DESIGN parameter in DBASEWIN.INI. To do so, either use the SET command to specify the setting interactively, or enter the DESIGN parameter directly in DBASEWIN.INI.

Description

When SET DESIGN is ON, dBASE lets you use CREATE and MODIFY commands to create and modify tables, forms, labels, reports, text, and queries. To prevent users of your applications from creating and modifying these types of files, issue SET DESIGN OFF in your programs.

If you issue SET DESIGN ON or OFF in a subroutine, the setting is effective only during execution of that subroutine.

Example

The default setting of SET DESIGN is usually ON. In this example the default setting in dBASEWIN.INI is OFF so that a user cannot use CREATE and MODIFY:

```
* In dBASEWIN.INI
[OnOffCommandSettings]
design =OFF
```

Portability

Not supported in dBASE III PLUS.

See Also

CREATE, CREATE FORM, CREATE LABEL, CREATE REPORT, MODIFY COMMAND, MODIFY FILE, MODIFY STRUCTURE

SET DEVELOPMENT

Programs

Determines whether dBASE automatically compiles a program, procedure, or format file when you change the file and then execute it or open it for execution.

Syntax

SET DEVELOPMENT ON | off

Default

The default for SET DEVELOPMENT is ON. To change the default, set the DEVELOPMENT parameter in DBASEWIN.INI. To do so, either use the SET command to specify the "Ensure Compilation" setting interactively, or enter the DEVELOPMENT parameter directly in DBASEWIN.INI.

Description

When SET DEVELOPMENT is ON and you execute a program file with DO, or open a procedure or format file, dBASE compares the time and date stamp of the source file and the compiled file. If the source file has a later time and date stamp than the compiled file, dBASE recompiles the file.

When SET DEVELOPMENT is ON and you change a source program, procedure, or format file with MODIFY COMMAND, dBASE erases the corresponding compiled file. When you then execute the program or open the procedure or format file, dBASE recompiles it.

When SET DEVELOPMENT is OFF, dBASE doesn't compare time and date stamps, and executes or opens existing compiled program, procedure, or format files. When you modify a source file and then open or execute it, dBASE first looks for a compiled file in memory and executes it if found. If no compiled file is in memory, dBASE looks for a

S

compiled disk file and executes it if found. If no compiled file is found, dBASE compiles the file.

When you DO a program, open a procedure file with SET PROCEDURE, or open a format file with SET FORMAT, dBASE always looks for, opens, and executes a compiled file. Therefore, if dBASE can't find a compiled version of a source file when you execute or open the source, dBASE compiles the file regardless of the SET DEVELOPMENT setting.

During program development, when you're editing files often, you should turn SET DEVELOPMENT ON. This ensures that you're always executing an up-to-date compiled file.

Turn SET DEVELOPMENT OFF when you no longer plan to change any source code. Turning SET DEVELOPMENT OFF speeds up program execution because dBASE doesn't have to check time and date stamps. You might want to set the DEVELOPMENT parameter to OFF in the DBASEWIN.INI file you distribute with your compiled code.

Portability

Not supported in dBASE III PLUS.

See Also

CLEAR PROGRAM, COMPILE, DO, SET PROCEDURE

SET DEVICE

Input/Output

Directs the output of @...SAY...GET commands, or *non-streaming output*, to the results pane of the Command window or current dBASE IV window, the printer, or a file. This command is supported primarily for compatibility with dBASE IV. In *Visual* dBASE, use SET PRINTER to direct *streaming output* to the printer or a file.

For complete syntax information on SET DEVICE, see online Help. For more information on streaming output, see Chapter 24 in the *Programmer's Guide*.

SET DIRECTORY

Disk and file utilities

Changes the current working drive or directory.

Syntax

SET DIRECTORY TO
[<path>]

<path> A character expression indicating the default path. To specify a root path, start <path> with a backslash (\) or the root directory (as with C:\). If <path> doesn't begin with \ or the root directory, dBASE begins the path with the current directory.

Default

The default working directory is the same as the directory returned by HOME(); that is, the directory containing the dBASE program files. You can change the default in Windows or in dBASE. In Windows, if you specify a Working Directory when you create or modify the dBASE icon properties, dBASE uses that directory as its default.

You can also update the DIRECTORY setting in DBASEWIN.INI. To do so, either use the SET command to specify the setting interactively, or enter the DIRECTORY parameter directly in DBASEWIN.INI.

A directory in the DBASEWIN.INI file overrides any in the dBASE icons Working Directory field.

Description

SET DIRECTORY lets you change the current working drive and directory. Use SET DIRECTORY to change the current working directory to any valid drive and path. The current directory appears in the Navigator window.

dBASE returns an error message if you try to SET DIRECTORY to a drive that doesn't exist or isn't ready. If you're unsure whether a drive is valid and ready for use, issue VALIDDRIVE() before using SET DIRECTORY.

SET DIRECTORY TO .. (two periods) changes the directory to the directory one level above the current directory. SET DIRECTORY TO without the option *<path>* sets the directory to the dBASE icons Working Directory, if there is one specified, or else to the HOME() directory.

Another way to access files on different directories is with the command SET PATH. You can specify one or more search paths, and dBASE uses these paths to locate files not on the current directory. Use SET PATH when an application's files are in several directories. You can also use SET DEFAULT to specify the drive on which dBASE searches for and stores files.

If you change the drive with SET DIRECTORY, the default drive (SET DEFAULT) is also set to the new drive.

SET DIRECTORY works like CD, except CD with no argument causes dBASE to display the current drive and directory. See CD for details on path statements.

Example

The following example saves the current directory, then uses SET DIRECTORY to change directories several times and examine the files in these subdirectories. Finally, the example returns to the original directory:

```
* This example assumes that directories were created by:
* MD D:\Project
* MD D:\Project\Programs
* MD D:\Project\Data
* MD C:\Editor
*
Olddir=SET("DIRECTORY")           && Original Directory
SET DIRECTORY TO D:\Project\Data
DIR                               && Displays DBFs
```

S

SET DISPLAY

```
SET DIRECTORY TO D:\Project\Programs
DIR *.prg                                && Display the programs
SET DIRECTORY TO ..
* moves from D:\Project\Programs to D:\Project
DIR *.*
SET DIRECTORY TO C:\Editor               && Change to C:\Editor
DIR *.doc
SET DIRECTORY TO &Olddir                 && Return to original directory
```

Portability

Not supported in dBASE III PLUS.

See Also

CD, HOME(), MKDIR, SET(), SET PATH, VALIDDRIVE()

SET DISPLAY

Environment

Makes the results pane of the Command window correspond to the specified DOS display mode. This command is supported primarily for compatibility with dBASE IV. You don't need to issue this command for new applications running only in Windows.

For more information about SET DISPLAY, see online Help.

SET ECHO

Error handling and debugging

Opens the dBASE Debugger. This command is supported primarily for compatibility with dBASE IV. In *Visual* dBASE, use DEBUG to open the debugger.

For complete syntax information on SET ECHO, see online Help.

SET EDITOR

Environment

Specifies the text editor to use when creating and editing programs and text files.

Syntax

```
SET EDITOR TO
[<expC>]
```

<expC> The expression you would enter at the DOS prompt or as the Windows command line to start the editor, usually the name of the editor's executable file (.EXE) or a Windows .PIF file. If <expC> doesn't include the file's full path name, dBASE looks for the file in the current directory, then in the DOS path.

Default

The default for SET EDITOR is the *Visual* dBASE internal Text Editor. To specify a different default editor, set the EDITOR parameter in DBASEWIN.INI. To do so, either

use the SET command to specify the setting interactively, or enter the EDITOR parameter directly in DBASEWIN.INI.

Description

Use SET EDITOR to specify an editor other than the default dBASE Text Editor to use when creating or editing text files. The file name you specify can be any text editor that produces standard ASCII text files. The specified editor opens when you issue CREATE/MODIFY FILE or CREATE/MODIFY COMMAND. If you issue SET EDITOR TO without a file name for *<expC>*, dBASE returns to the default editor.

You can use SET EDITOR to specify a .PIF file, which is a Windows file that controls the Windows environment for a DOS application, or a Windows .EXE file. Start the DOS editor by running the .PIF file rather than the .EXE. When you issue commands that run in a DOS window, dBASE loads COMMAND.COM using DBASEWIN.PIF in the _dbwinhome directory. You can use the Windows PIF Editor to customize the settings in DBASEWIN.PIF. For more information about .PIF files, see your Windows documentation. If there is not enough memory available to access an external editor, dBASE returns an "Unable to execute DOS" error message.

If the text editor you specify is already in use when you open a memo or file for editing, a second instance of the editor starts.

Example

The following example changes the default editor to Brief, to Write, the Windows editor and back to the dBASE editor:

```
SET EDITOR TO "c:\brief\b"
* now c:\brief does not need to be in the path statements
* MODIFY COMMAND now accesses Brief.
MODIFY COMMAND TEMP.PRG
SET EDITOR TO
MODIFY COMMAND TEMP
* Reverts to dBASE editor
```

You might not have sufficient RAM to access an external editor in which case dBASE gives an "Unable to execute DOS" error message.

Portability

Not supported in dBASE IV or dBASE III PLUS.

See Also

_dbwinhome, MEMORY(), MODIFY COMMAND, MODIFY FILE

SET ENCRYPTION

Establishes whether a newly created dBASE table is encrypted if PROTECT is used.

Syntax

SET ENCRYPTION ON | off

Default

The default for SET ENCRYPTION is ON.

Description

This command determines whether copied dBASE tables (that is, tables created through the COPY, JOIN, and TOTAL commands) are created as encrypted tables. An encrypted table contains data encrypted into another form to hide the contents of the original table. An encrypted table can only be read after the encryption has been deciphered or copied to another table in decrypted form.

To access an encrypted table, you must enter a valid user name, group name, and password after the login screen prompts. Your authorization and access level determine whether you can or cannot copy an encrypted table. After you access the table, SET ENCRYPTION OFF to copy the table 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 dBASE (.DBF) tables. Encryption works only with PROTECT. If you do not enter dBASE or access the table through the log-in screen, you will not be able to use encrypted tables.

All encrypted tables used concurrently in an application must have the same group name.

Encrypted tables cannot be JOINed with unencrypted tables. Make both tables either encrypted or unencrypted before JOINing them.

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

See also

COPY TO, PROTECT, SET()

SET ERROR

Error handling and debugging

Specifies one character expression to precede error messages and another one to follow them.

Syntax

SET ERROR TO

[<preceding expC> [, <following expC>]]

<preceding expC> An expression of up to 33 characters to precede error messages. dBASE ignores any characters after 33.

<following expC> An expression of up to 33 characters to follow error messages. dBASE ignores any characters after 33. If you want to specify a value for <following expC>, you must also specify a value or empty string ("") for <preceding expC>.

Default

The default for the message that precedes error messages is "Error: ". The default for the message that follows error messages is an empty string. To change the default, set the ERROR parameter in DBASEWIN.INI, using the following format:

```
ERROR = <preceding expC> [, <following expC>]
```

Description

Use SET ERROR to customize the beginnings and endings of run-time error messages. SET ERROR TO without an argument resets the beginnings and endings to the default values.

SET ERROR is similar to ON ERROR; both can be used to customize error messages. SET ERROR, however, can only specify expressions to precede and follow a standard dBASE error message, while ON ERROR can specify the message itself. Also unlike ON ERROR, SET ERROR can't call a procedure that carries out a series of commands.

Example

Use SET ERROR to customize error messages.

```
SET ERROR TO "Oops! - ", " - Please fix this."
? "a" = 1                                && generate a runtime error
SET ERROR TO
```

Portability

Not supported in dBASE IV or dBASE III PLUS.

See Also

ERROR(), MESSAGE(), ON ERROR

SET ESCAPE

Specifies whether pressing *Esc* interrupts program execution.

Syntax

SET ESCAPE ON | off

Default

The default for SET ESCAPE is ON. To change the default, set the ESCAPE parameter in DBASEWIN.INI.

Description

When SET ESCAPE is ON, pressing *Esc* interrupts program execution. Use SET ESCAPE OFF in a program to prevent unexpected user interruptions and command file termination during the execution of commands such as INDEX, PACK, and COPY.

Note Use SET ESCAPE OFF only in tested programs. If SET ESCAPE is OFF and you have not used ON KEY or to designate another key that interrupts programs, you can interrupt program execution only by rebooting your computer. Rebooting your computer to interrupt program execution, in turn, can cause data loss.

Regardless of whether SET ESCAPE is ON or OFF, pressing *Esc* always interrupts the processing of commands that pause for input from the keyboard, including ACCEPT, BROWSE, EDIT, INPUT, and READ.

Example

In the following example there is a bug in the subroutine WontWork. The instructions and the loop test do not correspond. With SET ESCAPE ON, you can press the Escape key to interrupt the program. You can then choose SUSPEND to examine the variable MORE or to follow the program through the loop:

```
SET ESCAPE ON
DO WontWork

PROCEDURE WontWork
More = ""
DO WHILE More <> "X"                                && should be Upper(More)<>"X"
? "Beginning the loop"
* ...
WAIT "Enter E to exit the loop" TO More
ENDDO
```

See Also

CLEAR TYPEAHEAD, INKEY(), ON ERROR, ON ESCAPE, ON KEY, READKEY(), SET KEY

SET EXACT

String data

Establishes the rules dBASE uses to determine whether two character strings are equal.

Syntax

SET EXACT on | OFF

Default

The default for SET EXACT is OFF. To change the default, set the EXACT parameter in DBASEWIN.INI. To do so, either use the SET command to specify the setting interactively, or enter the EXACT parameter directly in DBASEWIN.INI.

Description

When comparing two strings, dBASE compares the first character of the string on the right of the equal sign with the first character of the string on the left, then the second character of the right string to the second of the left string, and so on until two characters don't match or all characters of the right string have been examined.

If there are more characters in the string on the right, dBASE always returns .F. when the strings are compared. Therefore, if you know one string is longer than the other, put the longer string on the left side of the comparison operator.

When SET EXACT is ON, dBASE recognizes two character strings as equal when they are identical. When SET EXACT is OFF, dBASE recognizes two character strings as equal if all the characters in the string to the right of the comparison operator (=) match the beginning characters in the string to the left.

For example, ? "abc"="abcdef" returns .F. whether SET EXACT is ON or OFF, because all the characters in the string on the right do not match the beginning characters in the string on the left. (There are more characters on the right than on the left.)

When SET EXACT is OFF, ? "abcdef"="abc" returns .T. because the characters in the string on the right *do* match the beginning characters in the string on the left. When SET EXACT is ON, ? "abcdef"="abc" returns .F., because the strings are not identical.

When SET EXACT is OFF and the right string is an empty string (""), dBASE returns .T. when any string is compared to the empty string. When SET EXACT is ON and the right string is an empty string, dBASE returns .T. only when another empty string is compared to it.

To force an exact comparison whether SET EXACT is ON or OFF, use == instead of =, as shown in the following example.

```
SET EXACT OFF
? "abcd" = "abc"           && returns .T.
? "abcd" == "abc"         && returns .F.
```

In language drivers that have *primary* and *secondary weights* for characters (not U.S. language drivers but most others), dBASE compares characters by their primary weights when SET EXACT is OFF and by their secondary weights when SET EXACT is ON. For example, when SET EXACT is OFF, and the current language driver is German, "drücker" and "drucker" are equal.

S

SET EXACT affects all commands and functions that involve a string comparison, including FIND, SEEK, SEEK(), LOCATE, LOOKUP(), and any command executed with the FOR or WHILE option. For FIND and SEEK, dBASE treats the character string you specify as the right string of the comparison and the index file's key field as the left string.

Example

The following example uses SET EXACT to control how close to check for a match between two strings:

```
SET EXACT ON
? "Will" = "William"           && Returns .F.
? "William" = "Will"          && Returns .F.
SET EXACT OFF
? "Will" = "William"          && Returns .F.
? "William" = "Will"          && Returns .T.
```

The second example uses SET EXACT to demonstrate the difference between locating a memory variable with the SEEK command when EXACT is set OFF or ON:

```
SET TALK OFF
SET SAFETY OFF
SET EXACT OFF
USE Clients EXCLUSIVE
INDEX ON UPPER(Contact) TAG Contact
Lookup="Martin"
SEEK UPPER(Lookup)
IF FOUND( )
    DISPLAY FIELDS Company,Contact
ELSE
    ? "With EXACT SET OFF, the record was not found"
ENDIF

SET EXACT ON
SEEK UPPER(Lookup)
IF FOUND( )
    DISPLAY FIELDS Company,Contact
ELSE
    ? "With EXACT SET ON, the record was not found"
ENDIF
SET TALK ON
SET SAFETY ON
CLOSE DATABASES
```

Portability

In dBASE IV, constants are evaluated during compilation and not again during program execution. ? "abc"="a" returns .T. if SET EXACT is OFF when the program is compiled (and wasn't explicitly set in the program). If you later SET EXACT to ON in the Command window, ? "abc"="a" still returns .T. in the program.

In *Visual* dBASE, constants are evaluated during program execution; ? "abc"="a" returns .T. or .F. depending on the setting of SET EXACT when the program is run.

See Also

CHARSET(), FIND, LDRIVER(), LOCATE, LOOKUP(), SEEK, SEEK(),
SET LDCHECK

SET EXCLUSIVE

Shared data

Controls whether dBASE opens tables and their associated index and memo files in exclusive or shared mode.

Syntax

SET EXCLUSIVE on | OFF

Default

The default for SET EXCLUSIVE is OFF. To change the default, set the EXCLUSIVE parameter in DBASEWIN.INI. To do so, either use the SET command to specify the setting interactively, or enter the EXCLUSIVE parameter directly in DBASEWIN.INI.

Description

When you issue SET EXCLUSIVE ON, subsequent tables you open—and their associated indexes and memos—are in *exclusive mode*, unless you open them with USE...SHARED. When you open a table in exclusive mode, other users can't open, view, or change the file or any of its associated index and memo files. If you try to open a table that another user has opened in exclusive mode, dBASE displays an error message.

SET EXCLUSIVE OFF causes subsequent tables you open—and their associated indexes and memos—to be in *shared mode*, unless you open them with USE...EXCLUSIVE. If a table in shared mode is in a shared network directory, other users on the network with access to the directory can open, view, and change the file and any of its associated index and memo files.

If you use SET INDEX and the table is open in exclusive mode, dBASE opens the index in exclusive mode. If the table is open in shared mode by way of an overriding USE...SHARED, dBASE opens the index in the mode specified by USE.

An index created with INDEX is opened in exclusive mode, regardless of whether the table is opened in shared or exclusive mode and regardless of the SET EXCLUSIVE setting. After creating an index, you can open the index in shared mode with USE...INDEX...SHARED or by issuing SET EXCLUSIVE OFF followed by SET INDEX TO.

The following commands require the exclusive use of a table with either SET EXCLUSIVE ON or USE...EXCLUSIVE:

- CONVERT
- COPY INDEXES
- DELETE TAG
- INDEX...TAG
- INSERT
- INSERT AUTOMEM

SET FIELDS

- INSERT BLANK
- MODIFY STRUCTURE
- PACK
- REINDEX
- ZAP

Example

The following example uses the Company table with SET EXCLUSIVE ON so that a new index can be created. Once the index is created Company is reopened with exclusive off:

```
CLOSE ALL
SET EXCLUSIVE ON
USE Company
INDEX ON CompCode+Company TAG CompComp
SET EXCLUSIVE OFF
* New USE commands will not be exclusive,
* but Company is currently exclusive
USE Company ORDER CompComp
* This releases the exclusive lock on Company
```

Portability

Not supported in dBASE III PLUS.

See Also

FLOCK(), INDEX, RLOCK(), SET INDEX, SET LOCK, USE

SET FIELDS

Fields and records

Defines a group of fields to access in a table.

Syntax

```
SET FIELDS TO
[<field list> | ALL [LIKE <skeleton 1>] [EXCEPT <skeleton 2>]]
```

SET FIELDS on | OFF

<field list> | ALL [LIKE <skeleton 1> | EXCEPT <skeleton 2>] The fields list accessible in subsequent table operations. The fields list may include fields from tables open in all work areas and may also include read-only calculated fields. The following table provides a description of SET FIELDS TO options:

Option	Description
ALL	Makes all fields accessible in all work areas
LIKE <skeleton 1>	Makes all fields whose names match <skeleton 1> accessible in all work areas

Option	Description
EXCEPT <skeleton 2>	Makes all fields except those whose names match <skeleton 2> accessible in all work areas
LIKE <skeleton 1> EXCEPT <skeleton 2>	Makes all fields whose names are like <skeleton 1> except those whose names match <skeleton 2> accessible in all work areas

Default

The default for SET FIELDS is OFF; however, specifying a fields list with SET FIELDS TO automatically sets FIELDS ON.

Description

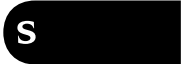
The SET FIELDS command defines a list of fields that can be accessed in one or more tables. You can specify fields in other work areas by specifying fields prefixed by an alias name. The list of fields specified with the SET FIELDS command is not active unless you set FIELDS ON. When SET FIELDS is OFF, all fields in a table are accessible.

SET FIELDS is automatically set ON when you define a fields list with SET FIELDS TO <field list>. You might, however, want to use SET FIELDS OFF to have all fields available in certain circumstances and then switch back later to the restricted fields list you have specified with the SET FIELDS TO command. If you SET FIELDS ON without using the SET FIELDS TO <field list> command, no fields are accessible.

SET FIELDS TO with no parameters clears the fields list, makes all fields accessible, and sets FIELDS OFF. (The CLEAR FIELDS command also clears the fields list.)

SET FIELDS affects the following commands, allowing access to only the fields specified in <field list> when SET FIELDS is set ON:

APPEND	COPY TO ARRAY
AVERAGE	DISPLAY
BLANK	EDIT
BROWSE	JOIN
CALCULATE	LIST
CHANGE	SUM
CREATE MODIFY VIEW	SET CARRY
COPY	TOTAL
COPY STRUCTURE	DISPLAY



SET FIELDS TO doesn't affect the following commands:

INDEX	SET FILTER
LOCATE	SET RELATION

The fields list specified with SET FIELDS TO can include both table field names and calculated fields. The /R option provides a setting to specify read-only access to table fields, for example:

```
Salary/R, Hours/R
```

To specify a calculated field, you can specify any valid dBASE expression. For example,

```
Gross_pay = 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. SET FIELDS TO ALL includes all fields of the current table.

To add to a previously specified fields list, use SET FIELDS TO again, with additional fields. For example, if you specify SET FIELDS TO Field1, and then SET FIELDS TO Field2, both Field1 and Field2 are in the fields list.

Example

The following example uses SET FIELDS ON or OFF to alternate between a fields list declared with SET FIELDS TO and a table in the current work area:

```
SELECT 1
USE Contact Order CompCode
SELECT 2
USE Company
SET RELATION TO CompCode INTO Contact
SET FIELDS TO Contact->CompCode, Company->Company, ;
    Contact->Contact, Company->Phone
SET FIELDS OFF
BROWSE                                && Company displayed in Browse
SET FIELDS ON
BROWSE                                && Fields list displayed
CLEAR FIELDS
```

See Also

CLEAR FIELDS, SET(), SET CARRY, SET RELATION

SET FILTER

Table organization

Sets a condition to limit records processed by dBASE commands.

Syntax

```
SET FILTER TO
[<condition>] | [FILE <filename> | ? | <filename skeleton>]
```

<condition> The condition that records must meet to be processed by subsequent dBASE commands.

FILE <filename>] | ? | <filename skeleton> The query file that specifies the filter condition. SET FILTER TO FILE ? and SET FILTER TO FILE <filename skeleton> display a dialog box, in

which you can select a file. If you specify a file without including its path, *Visual* dBASE looks for the file in the current directory, then in the path you specify with SET PATH. If you specify a file without including its extension, *Visual* dBASE assumes a .QRY extension.

Description

Use SET FILTER to view or work with only a portion of the records of a table. SET FILTER TO with no condition or file specified removes the current filter condition, causing *Visual* dBASE to process all records in the table. You can set up a separate filter condition for each table you open in a different work area.

SET FILTER TO <condition> sets up a filter based on a valid dBASE expression. The condition can filter records in the current table based on an expression including all data types except memo. For example, you can specify SET FILTER TO Lastname = "Jones" to filter records in a character field, SET FILTER TO Departure > {01/01/94} to filter records based on a date field. You can also combine conditions, for example, specifying SET FILTER TO Lastname = "Jones" .AND. Departure > {01/01/94}.

SET FILTER TO FILE <filename> obtains a filter condition from a dBASE III PLUS query (.QRY) file. You cannot use a dBASE IV or *Visual* dBASE .QBE file with this command.

Filters specified with the SET FILTER command aren't activated until after the record pointer is moved within a table (for example, using the GO TOP or SKIP command). All commands that require a table in use, such as AVERAGE, BROWSE, EDIT, and REPORT, can use conditions specified by SET FILTER.

A filter condition doesn't affect commands that identify records directly by number, such as GOTO <expN>. GOTO <expN> can move the pointer to a record that doesn't meet the filter condition; however, you cannot display that record (for example, using the EDIT command).

Example

The following example uses SET FILTER to restrict the following BROWSE command to Clients in California:

```
USE Clients
SET FILTER TO State_Prov = "CA"
BROWSE FIELDS Company, Contact, City, ;
    State_Prov, Cuisine
SET FILTER TO                                && release the filter
```

The following example uses SET FILTER to select from a RELATED table:

```
CLOSE DATA
USE Company EXCLUSIVE
INDEX ON Compcode TAG Compcode
SELE 2
USE Contact EXCLUSIVE
INDEX ON Contact TAG Compcode
SET RELATION TO Compcode INTO Company
SET FILTER TO Company->State_Prov="CA"
DISPLAY ALL Contact, Compcode,;
    Company->Company, Company->State_prov
```

Only contacts of companies in California are displayed.

See Also

CREATE QUERY, MODIFY QUERY, SET(), SET DELETED, SET KEY

SET FORMAT

Input/Output

Opens the specified format file in the selected work area, closing any currently open format file. Also compiles the format file if it hasn't already been compiled. This command is supported primarily for compatibility with dBASE IV. In *Visual dBASE*, use DEFINE to create *forms*, which are used instead of dBASE IV windows or format files.

For complete syntax information on SET FORMAT, see online Help. For information about defining forms, see the Forms chapters in the *User's Guide*.

SET FULLPATH

Environment

Specifies whether functions that return file names return the full path with the file name.

Syntax

SET FULLPATH ON | OFF

Default

The default for SET FULLPATH is OFF. To change the default, set the FULLPATH parameter in DBASEWIN.INI. To do so, either use the SET command to specify the setting interactively, or enter the FULLPATH parameter directly in DBASEWIN.INI.

Description

Use SET FULLPATH ON when you need to have functions such as CATALOG(), DBF(), MDX(), and NDX() return a file name with its full path. For example, if you are using tables in SQL databases, or tables located in different directories, issue SET FULLPATH ON to ensure that DBF() returns the full file name for use in subsequent commands.

Example

The following example obtains the name of the current table using both FULLPATH ON and FULLPATH OFF:

```
USE COMPANY
SET FULLPATH ON
? DBF()
* FULLPATH ON returns G:\VISUALDB\EXAMPLES\COMPANY.DBF
SET FULLPATH OFF
? DBF()
* FULLPATH OFF returns G:COMPANY.DBF
```

Portability

Not supported in dBASE III PLUS. If you want file names returned in the same format as they are in dBASE III PLUS, issue SET FULLPATH OFF.

See Also

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

SET FUNCTION

Keyboard and mouse events

Assigns a command or expression to a function key or to a combination of the *Ctrl* (control) key or the *Shift* key and a function key.

Syntax

SET FUNCTION <key> TO <expC>

<key> A function key number, function key name, or character expression of a function key name—for example, 3, F3, or "F3". Specify a character expression for <key> to assign a key combination using the Ctrl or Shift key with a function key. Type "CTRL+" or "SHIFT+" in uppercase or lowercase and then a function key name—for example, "shift+F5" or "Ctrl+f3". For compatibility with dBASE IV, you can use a hyphen in place of the plus sign. You can't combine Ctrl and Shift, such as "Ctrl+Shift+F3".

<expC> A dBASE command, function, user-defined function (UDF), or any character string. Follow a command with a semicolon (;) to execute the command immediately when you press <key>. You can execute more than one command by separating each command in the list with a semicolon.

Default

The following function key settings are in effect when dBASE starts:

Key	Command	Key	Command
F1	HELP;	F7	DISPLAY MEMORY;
F3	LIST;	F8	DISPLAY;
F4	DIR;	F9	APPEND;
F5	DISPLAY STRUCTURE;	F10	Activates the menu
F6	DISPLAY STATUS;		

Description

Use SET FUNCTION to assign commands to a function key or a function key combination with *Ctrl* or *Shift*. You can also use SET FUNCTION to assign any character expression to a function key. When you're in the Command window and press the key or key combination, <expC> appears at the cursor. If <expC> ends with a semicolon, dBASE executes it immediately. If <expC> doesn't end with a semicolon (;), you can edit it.



Note *F2* is reserved for toggling between views while in the Browse window. You can program it, but it won't be recognized when in the Browse window. You can't program *F10*, or any combination using *F11* or *F12*.

In forms, expressions assigned with SET FUNCTION are recognized only when you are in Entry Fields.

For more information on programming function keys, as well as on programming other keys, see ON KEY.

Example

The following example issues the SET command when the user presses a function key. This activates the PROPERTIES | ENVIRONMENT menu which lets the user change any SET command defaults:

```
SET FUNCTION F9 TO "SET;"
```

SET FUNCTION can also be used to issue multiple commands as follows:

```
SET FUNCTION F8 TO "CLOSE ALL;CLEAR;"+;
"USE Clients;BROWSE;"
```

See Also

DISPLAY STATUS, FKLABEL(), FKMAX(), HELP, INKEY(), ON KEY

SET HEADINGS

Input/Output

Controls the display of field names in the output of AVERAGE, DISPLAY, LIST, and SUM.

Syntax

SET HEADINGS ON | off

Default

The default for SET HEADINGS is ON. To change the default, set the HEADINGS parameter in DBASEWIN.INI. To do so, either use the SET command to specify the setting interactively, or enter the HEADINGS parameter directly in DBASEWIN.INI.

Description

When SET HEADINGS is ON, the output of AVERAGE, DISPLAY, LIST, and SUM includes a heading identifying the fields of the table(s). Issue SET HEADINGS OFF before issuing AVERAGE, DISPLAY, LIST, or SUM to view output data without field-name headings.

Example

The following example uses DISPLAY with SET HEADING ON to show the field names and SET HEADING OFF to hide the field names:

```

USE Clients
SET HEADING OFF
? "Heading off"
DISPLAY Client_id, Company
SET HEADING ON
? "Heading on"
DISPLAY Client_id, Company
*
* Heading off
*      1  A8513 A Beck Pertamina
*
* Heading on
* Record#  Client_id Company
*      1  A8513      A Beck Pertamina

```

Portability

In dBASE III PLUS, the command is SET HEADING (no final s).

In dBASE IV, SET HEADINGS also affects the results of the TYPE command, determining whether the display of the typed file includes a heading specifying the file name and date. In *Visual* dBASE, TYPE does not include this heading, regardless of the SET HEADINGS setting.

See Also

AVERAGE, DISPLAY, LIST, SUM, TYPE

SET HELP

Windows programming

Determines which Help file (.HLP) the dBASE Help system uses.

Syntax

```

SET HELP TO
[<help filename> | ? | <help filename skeleton>]

```

<help filename> | ? | <help filename skeleton> Identifies the Help file to activate. ? and <filename skeleton> display a dialog box, from which you can select a file. If you specify a file without including its extension, dBASE assumes .HLP.

Description

Use SET HELP TO to specify which Help file to use when the dBASE Help system is activated.

The Help file is opened automatically when you start dBASE if you place the file in the dBASE home directory.

SET HELP TO closes any open Help file before it opens a new file.

Example

To display a dialogue box to select from available Help files, issue the following command:

```
SET HELP TO ?    && or optionally
SET HELP TO *.HLP
```

To set Help to your own tailored help file:

```
SET HELP TO MyHlp.HLP
```

To set the Help file back to the *Visual* dBASE default:

```
SET HELP TO \VISUALDB\BIN\DBASEWIN.HLP
```

Portability

Not supported in dBASE IV or dBASE III PLUS.

See Also

HELP, HelpFile, HelpID, SET TOPIC

SET IBLOCK

Table organization

Changes the default block size used for new .MDX files.

Syntax

```
SET IBLOCK TO <expN>
```

<expN> A number from 1 to 63 that sets the size of index blocks allocated to new .MDX files. The default value is 1. (The actual size in bytes is the number you specify multiplied by 512 bytes; however, the minimum size of a block is 1024 bytes.) To change the default, update the IBLOCK setting in DBASEWIN.INI. To do so, either use the SET command to specify the setting interactively, or enter the IBLOCK parameter directly in DBASEWIN.INI.

Description

Use SET IBLOCK to change the size of blocks in which *Visual* dBASE stores .MDX files on disk to improve the performance and efficiency of indexes. You can specify a block size from 1024 bytes to approximately 32K. The IBLOCK setting overrides any previous block size defined by the SET BLOCKSIZE command or specified in the DBASEWIN.INI file. After the block size has been changed, new .MDX index files are created with the new block size.

Multiple index (.MDX) files are composed of individual index blocks (or *nodes*). Nodes contain the value of keys corresponding to individual records and provide the information to locate the appropriate record for each key value. Since the IBLOCK setting determines the size of nodes, the setting also determines the number of key values that can fit into each node. When a single node can't contain all the key values in an index, *Visual* 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. If you increase the size of index blocks and create a new .MDX file, the new and larger leaf nodes contain more key values.

Whether you can improve performance by storing key values in larger or smaller nodes depends on several factors: the distribution of data, if tables are linked together, the length of 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 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 might not be sequential, *Visual* dBASE reads only one node at a time from the disk. Reading more than one node is usually inefficient, because typically the second node is not the next node in the sequential list.
- Once a node is read into memory, *Visual* 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 tables together, for example, with SET RELATION, performance is better if all the relevant nodes for the tables are in memory simultaneously. For example, if a large node for table B pushes out the previously read node for table A, *Visual* dBASE must find and read the table A node again from disk when the node for table A needs to be used again. If both nodes remain in memory, performance can be improved.
- When tables have many identical key values, *Visual* dBASE might have to store them in many nodes. In this situation, performance might be improved by increasing the node size so that *Visual* dBASE reads fewer nodes from 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, *Visual* dBASE 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

This example creates two .MDXs containing identical data but with different IBLOCK settings and consequently, different file sizes:

```
CLOSE DATA
DELETE FILE Co1.mdx
DELETE FILE Co2.mdx
* remove any previous .mdx
* CREATE THE MDXs
USE Company EXCLUSIVE
SET IBLOCK TO 2
INDEX ON CompCode TAG CompCode OF Co1
INDEX ON Company TAG Company OF Co1
INDEX ON City TAG City OF Co1
SET IBLOCK TO 20
INDEX ON CompCode TAG CompCode OF Co2
INDEX ON Company TAG Company OF Co2
INDEX ON City TAG City OF Co2
DIR CO?.MDX
```

Two .MDXs, Co1.MDX and Co2.MDX are created with different IBLOCK settings. CO1 and CO2 will have different file sizes because their block lengths are different.

Portability

Not supported in dBASE III PLUS.

See Also

COPY, COPY INDEXES, CREATE, MODIFY STRUCTURE, INDEX, REINDEX, REPLACE, SET(), SET BLOCKSIZE, SET MBLOCK

SET INDEX

Table organization

Opens index files for the current table. Not applicable for Paradox and SQL tables.

Syntax

```
SET INDEX TO
[<filename list> | ? | <filename skeleton>]
[ORDER [TAG]
  <filename 1> | <tag name> [OF
    <filename 2> | ? | <filename skeleton>]]
```

<filename list> | ? | <filename skeleton> Specifies the index files to open, including both single and multiple indexes. SET INDEX TO ? and SET INDEX TO <filename skeleton> display a dialog box, in which you select an index file. If you specify a file without including its path, *Visual* dBASE looks for the file in the current directory, then in the path you specify with SET PATH.

If you specify a file without including its extension, *Visual* dBASE assumes an .NDX extension for single index files and an .MDX extension for multiple index files. If you have an .NDX file and an .MDX file with the same name, to open the .NDX file, specify the .NDX extension.

ORDER [TAG] <filename 1> | <tag name> Specifies a master index that can be an .NDX file or a tag name contained within an .MDX index file.

OF <filename 2> | ? | <filename skeleton> Specifies a multiple index file containing <tag name>. OF ? and OF <filename skeleton> display a dialog box, in which you select a multiple index file. If you specify a file without including its path, *Visual* dBASE looks for the file in the current directory, then in the path you specify with SET PATH. If you specify a file without including its extension, *Visual* dBASE assumes a .MDX extension.

Description

Use SET INDEX to open the specified .NDX and .MDX files in the current work area that will be updated when changes are made to the associated table. Including an index file list when issuing USE...INDEX is equivalent to following the USE command with the SET INDEX command.

If the first index opened with SET INDEX is an .NDX file, that index becomes the master index unless you specify another master index with the ORDER option or the SET

ORDER command. If the first index opened with SET INDEX is an .MDX file and you don't specify the ORDER clause, no master index is defined, and records in the table appear in record number or *natural* order. To specify a master index for the current table, specify the ORDER option or use the SET ORDER command.

Before opening the indexes specified with the command, SET INDEX closes all open index files except those index files with tag names listed in the production .MDX file, the index file with the same name as the current table. Specifying SET INDEX TO without a list of indexes closes all open .NDX and .MDX files in the current work area, except for the production index file. You can also use the CLOSE INDEX command. All indexes, including the production .MDX file, are closed when you close the table.

The order in which you specify indexes with the SET INDEX command isn't necessarily the same as the order *Visual dBASE* uses. Open indexes for a specified work area are listed in the following order:

- 1 Any single index files are in the order you list them in *<filename list>*.
- 2 Indexes in the production .MDX file are in the order you list them in the .MDX file.
- 3 Indexes in other .MDX files you specify with USE...INDEX or the SET INDEX command are in the order you list them in the individual .MDX files.

The order of the open indexes remains the same until you specify another index order with the USE...INDEX or SET INDEX commands, or you issue an INDEX command. Use the order number as an argument with NDX(), TAG(), and KEY() to determine an index file name, tag name, or key expression.

Example

The following example uses SET INDEX to open index tags and set the controlling index:

```
SET SAFETY OFF                                && To avoid overwrite warning
USE Company EXCLUSIVE
INDEX ON Company TAG Company                  && TAG Company in Company.mdx
INDEX ON Compcode TAG Compcode
SET INDEX TO Company ORDER Company
BROWSE FIELDS Company, Compcode;
TITLE "Indexed by Company"
```

The following example uses SET INDEX to open .NDX index files and set the controlling index:

```
USE Company EXCLUSIVE
INDEX ON Company TO Company                    && Company.ndx
INDEX ON Compcode TO Compcode                  && Compcode.ndx
SET INDEX TO Compcode.ndx, Company.ndx
* These are .NDX indexes, not .MDX tags
BROWSE FIELDS Compcode, Company;
TITLE "Indexed by Compcode"
```

S

See Also

CLOSE..., INDEX, KEY(), MDX(), NDX(), ORDER(), REINDEX, SET ORDER, TAG(), TAGNO(), TAGCOUNT(), USE

SET INTENSITY

Environment

Determines whether dBASE displays variable and field data during data-entry operations in dBASE IV windows in reverse video (monochrome monitors) or in the colors specified for enhanced text with the SET COLOR or SET COLOR OF command (color monitors). SET INTENSITY is supported primarily for compatibility with dBASE IV. To define colors for Windows objects, use the ColorNormal and ColorHighlight properties.

For more information about SET INTENSITY, see online Help.

SET KEY

Keyboard and mouse events

Assigns a program or a procedure to a specified key or key combination. (The dBASE IV command SET KEY, which limits the records processed in a table to those whose key field value falls within a range, is now called SET KEY TO.)

Syntax

```
SET KEY <expN> | <expC> TO  
[<program name> | <procedure name>]
```

<expN> | <expC> The key you assign the program or procedure to. <expN> is the INKEY() value of the key or key combination. <expC> is a function specified by *F1* through *F9*, *Shift+F1* through *Shift+F10*, or *Ctrl+F1* through *Ctrl+F10*.

<program name> | <procedure name> The program or procedure you assign to the key or key combination.

Description

Use SET KEY to assign a program or a procedure to a key or a key combination. When the user presses the specified key or key combination, SET KEY causes dBASE to stop execution of the current program and execute the specified program or procedure. The original program resumes execution when the program or procedure that SET KEY executes is completed.

SET KEY is identical in function to ON KEY LABEL; only the syntax differs. If you use ON KEY LABEL and SET KEY to set the same key, dBASE executes the program or procedure specified by the most recently issued command. For more information, see ON KEY.

Example

The following example uses SET KEY to set a function key, an alt key, a control key and an alpha key as procedure calls:

```
SET PROCEDURE TO PROGRAM(1) ADDITIVE
SET KEY "F2" TO Open           && Do Open
SET KEY "ALT-F3" TO Close      && Do Close
SET KEY "CTRL-F4" TO New       && Do New
SET KEY "x" TO Exit            && Do Exit
```

```

KEYBOARD "x{F2}{ALT-F3}{ctrl-F4}"
* This will call each function in turn

* Warning: the letter X will call the Exit procedure and ;
* cannot be used for ordinary typing.
Now reset the keys:

SET KEY "F2" TO
SET KEY "ALT-F3" TO
SET KEY "CTRL-F4" TO
SET KEY "x" TO

PROCEDURE Open
? "In open procedure"
RETURN

PROCEDURE Close
? "In Close Procedure"
RETURN

PROCEDURE Exit
? "In Exit Procedure"
RETURN

PROCEDURE New
? "In New Procedure"
RETURN

```

Portability

Not supported in dBASE IV or dBASE III PLUS. However, ON KEY is supported in both.

See Also

ON KEY, SET ESCAPE, SET FUNCTION, SET KEY TO

SET KEY TO

Table organization

Limits the records processed in the current or specified table to those whose key field value falls within a range.

Syntax

```

SET KEY TO
[<exp list 1> |
  RANGE <exp 2> [,] |
    , <exp 3> |
    <exp 2>, <exp 3>
  [LOW <exp list 2> [,]
  [HIGH [,] <exp list 3>]
[EXCLUDE]
[IN <alias>]

```

S

<exp 1> | RANGE <exp2> [,] | ,<exp3> | <exp2>, <exp3> Specifies a condition that filters records. For Paradox and SQL tables, you can specify values (separated by commas) that match single or composite index key fields. The following table summarizes how SET KEY filters records in the master index:

Option	Description
<exp list1>	Searches for records whose index key values match <exp list 1>
RANGE <exp2> [,]	Searches for records whose index values are greater than or equal to <exp 2>
RANGE, <exp3>	Searches for records whose index values are less than or equal to <exp 3>
RANGE <exp2>, <exp3>	Searches for records whose index values are greater than or equal to <exp 2> and less than or equal to <exp3>

LOW <exp list 2> [,] Specifies that records whose index key values are less than the specified low values are not included with qualified records.

HIGH [,] <exp list 2> Specifies that records whose index key values are greater than the specified high values are not included with qualified records.

EXCLUDE When a range is included, specifies that records where index values match either the high or low value of the range are not included with qualified records.

IN <alias> Specifies a work area. You can enter a work area number (1 through 225), letter (A through J), or alias name. The work area letter or alias name must be enclosed in quotes.

Description

The SET KEY TO command limits the records that *Visual* dBASE processes in the current or specified table to those whose key field values fall within the specified range. Unless you specify the EXCLUDE keyword, *Visual* dBASE also includes key field values that match the high or low values of the specified range. SET KEY TO with no arguments removes any range of key values previously established for the current table with SET KEY.

The values specified in <exp 1>, <exp 2>, and <exp 3> must match the key expression of the master index. For example, if the index key is UPPER(Name), specify uppercase letters in the range expressions. In determining whether the specified range expressions match key field expressions, SET KEY TO follows the rules established by SET EXACT. The SET KEY TO range takes effect after you move the record pointer.

You cannot use SKIP or LOCATE to go to a record that falls outside the SET KEY range. However, you can use GO to move the record pointer to a record that is outside the SET KEY range. Also, commands such as REPLACE that process more than one record operate only on records that are within the SET KEY range. No records are processed when you specify a range that doesn't contain any records.

When you issue both SET KEY and SET FILTER for the same table, *Visual* dBASE processes only records that are within the SET KEY index range and that also meet the SET FILTER condition.

Example

The following example uses SET KEY four times to select records from the table:

```

USE Company EXCLUSIVE
INDEX ON State_Prov TAG State

SET KEY TO "CA"
LIST State_Prov
WAIT "only CA"

SET KEY TO RANGE , "IL"
LIST State_Prov
WAIT "up to Illinois"

SET KEY TO RANGE "GA",
LIST State_Prov
WAIT "Georgia and beyond"

SET KEY TO RANGE "GA", "IL"
LIST State_Prov
WAIT "From Georgia to Illinois only"

```

Portability

Not supported in dBASE III PLUS.

See Also

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

SET LDCHECK

Environment

Enables or disables language driver ID checking.

Syntax

SET LDCHECK ON | off

Default

The default for SET LDCHECK is ON. To change the default, set the LDCHECK parameter in DBASEWIN.INI.

Description

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

Language drivers determine the character set and sorting rules that dBASE uses, so if you create a dBASE table with one language driver and then use that file with a different language driver, some of the characters will appear incorrectly and you may get incorrect results when querying data.

For more information about working with language drivers, see Appendix C in the *Programmer's Guide*.

S

Example

See LDRIVER() for an alternative to SET LDCHECK ON.

Portability

Not supported in dBASE III PLUS.

See Also

CHARSET(), LDRIVER()

SET LD CONVERT

Environment

Determines whether data read from and written to character and memo fields is transliterated when the table character set does not match the global language driver.

Syntax

SET LD CONVERT ON | off

Default

The default for SET LD CONVERT is ON. To change the default, set the LD CONVERT parameter in DBASEWIN.INI.

Description

Use SET LD CONVERT to determine whether the contents of character and memo fields in tables created with a given language driver in effect, are converted to match the language driver in effect at the time the fields are read or written to.

Language drivers determine the character set and sorting rules that dBASE uses, so if you create a dBASE table with one language driver and then use that file with a different language driver, some of the characters will appear incorrectly and you may get incorrect results when querying data.

In general, SET LD CONVERT should be ON to insure that dBASE behaves as expected when using data created under disparate language drivers.

For more information about working with language drivers, see PG_CHARLANG.

Portability

Not supported in dBASE IV or dBASE III PLUS.

See Also

CHARSET(), LDRIVER(), SET LDCHECK

SET LIBRARY

Programs

Opens a dBASE program file (.PRG), making all procedures and user-defined functions (UDFs) in the file available for execution.

Syntax

SET LIBRARY TO

[<filename> | ? | <filename skeleton>]

<filename> | ? | <filename skeleton> The library file to open. The ? and <filename skeleton> options display a dialog box from which you can select a file. If you specify a file without including its path, dBASE looks for the file in the current directory, then in the path you specify with SET PATH. If you specify a file without including its extension, dBASE assumes .PRO (a compiled object file). If dBASE can't find a .PRO file, it looks for a .PRG file (a source file). If dBASE finds a .PRG file, it compiles it.

Description

SET LIBRARY complements SET PROCEDURE. Both commands open a file, allowing access to the procedures and UDFs the file contains. You can use SET LIBRARY TO without a file name to close an open library file.

To execute a procedure or UDF, dBASE must have access to the file containing it. When dBASE encounters a call to a procedure or UDF, it looks for the procedure or UDF in specific places in a specific order. One of the places dBASE looks is in a file opened with SET LIBRARY. See the DO command for an explanation of the search path and order dBASE uses.

For more information about working with a library file, see Chapter 4 in the *Programmer's Guide*.

Example

The following example uses SET LIBRARY to make additional procedures and functions from another file available to the main program:

```
** Rpt_Proc.PRG **
PROCEDURE Rpt_Head
line_cnt=1
DEFINE FORM MainForm FROM 0,0 TO 20,40
DEFINE TEXT line01 OF MainForm AT 1, 0 ;
    PROPERTY TEXT CENTER("Clients Database Report")
DEFINE TEXT line02 OF MainForm AT 2,19 ;
    PROPERTY TEXT ;
    CENTER("Run on " + DTOC(DATE()),40,"-")
DEFINE TEXT rpt_line OF MainForm ;
    AT line_cnt + 3,0 ;
    PROPERTY TEXT CENTER("Company",40)
DEFINE TEXT rpt_line2 OF MainForm ;
    AT line_cnt + 3,40 ;
    PROPERTY TEXT CENTER("Contact",40)
DEFINE TEXT line03 OF MainForm AT 2,19 ;
    PROPERTY TEXT CENTER(Next_Rpt(7),40,"-")
OPEN FORM MainForm
```

SET LOCK

```
FUNCTION Next_Rpt
PARAMETERS days
nextdate = "Next Report Due on " + DTOC( DATE() + days)
RETURN nextdate
```

The next part of the example is the master program file:

```
** Main.PRG **
SET PROCEDURE TO Main
SET LIBRARY TO Rpt_Proc
DO MainSetup
DO Rpt_Head
SET LIBRARY TO

PROCEDURE MainSetup
SET TALK OFF
SET ECHO OFF
CLEAR
SET DEVELOPMENT ON
RETURN
```

Portability

Not supported in dBASE III PLUS. The ? and <filename skeleton> options are not supported in dBASE IV.

In dBASE IV, a file opened with SET LIBRARY is always searched after a file opened with SET PROCEDURE. In *Visual* dBASE, the first file opened is the first one searched.

See Also

DO, FUNCTION, PROCEDURE, SET(), SET PROCEDURE

SET LOCK

Shared data

Determines whether dBASE attempts to lock a shared table during execution of certain commands that read the table but don't change its data.

Syntax

SET LOCK ON | off

Default

The default for SET LOCK is ON. To change the default, set the LOCK parameter in DBASEWIN.INI. To do so, either use the SET command to specify the setting interactively, or enter the LOCK parameter directly in DBASEWIN.INI.

Description

Issue SET LOCK OFF to disable automatic file locking for certain commands that only read a table. This lets other users change data in the file while you access it with read-only commands. For example, you might want to issue SET LOCK OFF before using AVERAGE if you don't expect other users to alter the data in the table you're using

significantly. Or, you might want to issue SET LOCK OFF before processing a range of records that other users aren't going to update.

The following commands automatically lock tables when SET LOCK is ON and don't lock tables when SET LOCK is OFF:

- AVERAGE
- CALCULATE
- COPY (source file)
- COPY MEMO
- COPY STRUCTURE
- COPY TO ARRAY
- COPY TO...STRUCTURE [EXTENDED] (source file)
- COUNT
- JOIN (both source files)
- LABEL FORM
- REPORT FORM
- SORT (source file)
- SUM
- TOTAL (source file)

dBASE continues to lock records and tables automatically for commands that let you change data whether SET LOCK is ON or OFF.

Example

This program demonstrates the use of SET LOCK in processing read-only data.

```
USE Company
SET LOCK OFF                                && Do not lock records during COUNT
COUNT FOR Company->Type = "VAR" TO nCount
CLEAR
? nCount
SET LOCK ON
```

Portability

Not supported in dBASE III PLUS.

See Also

FLOCK(), RLOCK()

Sets the width of the left border of a printed page.

Syntax

SET MARGIN TO <expN>

<expN> The column number at which to set the left margin. The valid range is 0 to 254, inclusive. You can specify a fractional number for <expN> to position output accurately with a proportional font.

Default

The default for SET MARGIN is 0. To change the default, set the MARGIN parameter in DBASEWIN.INI.

Description

Use SET MARGIN to adjust the printer offset for the left margin for all printed output. The margin established by SET MARGIN becomes the printer's column 0 position. SET MARGIN resets the value of the _ploffset system memory variable but doesn't affect the value of the _lmargin system memory variable.

Use SET MARGIN to adjust the position of text on the printed page according to the type of paper. For example, if you're printing to three-hole paper, you might need to increase the left border. You can also use SET MARGIN to compensate for the placement of paper in the printer. For example, if the paper is off-center in the printer, you can adjust the width of the left border to properly place the text.

When you direct output to the printer, dBASE maps each character according to the *coordinate plane*, a two-dimensional grid. The coordinate plane is divided into character cells whose widths depend on the value of _ppitch. See the table in the description of _ppitch, which lists _ppitch values. The height of each character cell is determined by the size of the font. For more information about the coordinate plane, see Chapter 16 in the *Programmer's Guide*.

If you change the value of SET MARGIN, dBASE takes the current value of _ppitch into consideration when calculating the cell width of the coordinate plane. This happens regardless of whether you're printing with proportional or monospaced fonts. If you issue ? without using the STYLE option and use only integer coordinates, dBASE uses a monospaced font, and all output appears exactly the same as in dBASE IV.

Example

This example displays the 10 digits with the default margin and then sets the margin to column 10 and displays the 10 digits this time indented:

```
SET PRINTER ON
SET MARGIN TO 0                && The default
? "1234567890"
SET MARGIN TO 10
? "1234567890"
? _ploffset
```

```

SET MARGIN TO 0
SET PRINTER OFF
CLOSE PRINTER

```

Produces:

```

* this displays as:
* 1234567890
*           1234567890
*           10
*
* _ploffset is set by SET MARGIN

```

See Also

_indent, _lmargin, _ploffset, _rmargin

SET MARK

Date and time data

Determines the character dBASE uses to separate the month, day, and year when it displays dates.

Syntax

```

SET MARK TO
[<expC>]

```

<expC> The single *date separator character*. You can specify more than one character for <expC>, but dBASE uses only the first one.

Default

The default for SET MARK is the separator character of the current date format in use by the system. dBASE uses the value set by SET DATE, DBASEWIN.INI, or the International option of the Windows Control Panel (in that order) to determine the current date format. That is, SET DATE settings override those in DBASEWIN.INI, and settings in DBASEWIN.INI override those in the Windows Control Panel.

To change the default separator character, set the MARK parameter in DBASEWIN.INI. To do so, either use the SET command to specify the setting interactively, or enter the MARK parameter directly in DBASEWIN.INI.

If you specify both DATE and MARK settings in DBASEWIN.INI, dBASE displays dates in the month/day/year format corresponding to the DATE setting, but uses the separator specified in the MARK setting. Once you've started dBASE, issuing SET DATE overrides any prior SET MARK setting, including one specified in DBASEWIN.INI.

Description

Use SET MARK to change the date separator from the default character. For example, if you issue SET DATE AMERICAN, the date separator character is a forward slash (/), and dBASE displays dates in MM/DD/YY format. However, if you specify SET MARK TO "." after issuing SET DATE AMERICAN, dBASE displays dates in the format

SET MBLOCK

MM.DD.YY. If you issue SET DATE AMERICAN again, the format returns to MM/DD/YY.

Issuing SET MARK TO without *<expC>* resets the date separator character to that of the current date format.

Example

The following example uses SET MARK to change the date separator when displaying date data:

```
date = {04/01/94}
SET CENTURY OFF
SET DATE AMERICAN
ENDIF
? date                                && Returns 04/01/94
SET CENTURY ON
? date                                && Returns 04/01/1994
SET MARK TO ", "
? date                                && Returns 04,01,1994
SET MARK TO "/"
? date                                && Returns 04/01/1994
SET MARK TO ":"? date                  && Returns 04:01:1994
SET MARK TO "-"
? date                                && Returns 04-01-1994
SET DATE AMERICAN
? date                                && Returns 04/01/1994
SET DATE USA
? date    && Returns 04-01-1994
```

Portability

Not supported in dBASE III PLUS.

See Also

DATE(), DMY(), MDY(), SET CENTURY, SET DATE

SET MBLOCK

Fields and records

Changes the default block size of new memo field (.DBT) files.

Syntax

SET MBLOCK TO *<expN>*

<expN> A number from 1 to 512 that sets the size of blocks used to store new memo (.DBT) files. (The actual size in bytes is the number you specify multiplied by 64.)

Default

The default value for SET MBLOCK is 8 (or 512 bytes). To change the default, update the MBLOCK setting in DBASEWIN.INI. To do so, either use the SET command to specify the setting interactively, or enter the MBLOCK parameter directly in DBASEWIN.INI.

Description

Use SET MBLOCK to change the size of blocks in which dBASE stores new memo field (.DBT) files on disk. You can specify a block size from 64 bytes to approximately 32K. The MBLOCK setting overrides any previous block size defined by the SET BLOCKSIZE command or specified in the DBASEWIN.INI file. After the block size has been changed, new memo .DBT files are created with the new block size. dBASE stores data in each memo field in a group made up of as many blocks as needed.

After the block size is changed, memo fields created with the COPY, CREATE, and MODIFY STRUCTURE commands have the new block size. To change the block size of an existing memo field file, use the SET BLOCKSIZE command to change the block size and then copy the table containing the associated memo field to a new file. The new file then has the new block size.

When the block sizes are large and the memo contents are small, memo (.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:

- You can allocate different block sizes for memo field and index data, whereas SET BLOCKSIZE requires the same block size for both. To allocate block sizes for index data, use SET IBLOCK.
- 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 uses SET MBLOCK to create another table that is a copy of Clients but has a memo blocksize of 256 bytes embedded in its structure versus the default of 512 bytes. This technique applies if memo entries are normally less than 256 bytes and you want to minimize wasted space in the .DBT file:

```
USE Clients
? SET("MBLOCK")                && Returns default of 8;each memo block = 512 bytes
SET MBLOCK TO 4
COPY TO Clients2
USE Clients2
? SET("MBLOCK")                && Returns 4
LIST FILES LIKE Clients*.DBT
* Note that Clients2.DBT is smaller than Clients.DBT
CLOSE DATABASES
```

S

Portability

Not supported in dBASE III PLUS.

See Also

CREATE, MODIFY STRUCTURE, REPLACE, SET(), SET BLOCKSIZE, SET IBLOCK

SET MEMOWIDTH

Sets the width of memo field display or output.

Syntax

SET MEMOWIDTH TO

[<expN>]

<expN> Specifies a number from 8 to 255 that sets the width of memo field display and output.

Default

The default memo width is set to 50.

Description

Use SET MEMOWIDTH to change the column width of memo fields during display and output. Memo fields can be displayed using the commands DISPLAY, LIST, ?, or ?. SET MEMOWIDTH doesn't affect the display of a memo field in the Text Editor. If the system memory variable variable_wrap is set to true (.T.), the system memory 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 specified, the _pcolno system memory variable is incremented by the @V value. This lets you change the appearance of the printed output of ? or ?? commands by using the @V function. When @V is equal to zero, memo fields wrap within the SET MEMOWIDTH width.

Example

The following example demonstrates the relationship between SET MEMOWIDTH and the value returned by MEMLINES(). The REPLACE command places a string in memo field Notes and SET MEMOWIDTH is used to alter the memo field length for output purposes:

```
SET MEMOWIDTH TO 30
USE CLIENTS
REPLACE Notes WITH "Mr. Jackson contacted us "+;
    "this date, 04/15/94 regarding his accident of "+;
    "04/10/94 involving a truck and his 1994 Lexus."
CLEAR
? "MEMOWIDTH", "MEMLINES" AT 20
? " 30", LTRIM(STR(MEMLINES(Notes))) AT 22
?
? Notes
?
SET MEMOWIDTH to 45
? "MEMOWIDTH", "MEMLINES" AT 20
? " 45", LTRIM(STR(MEMLINES(Notes))) AT 22
?
? Notes
```

See MEMLINES() for another example of SET MEMOWIDTH.

See Also
?, ??, DISPLAY, LIST, MEMLINES(), MLINE(), SET(), _lmargin, _rmargin, _wrap

SET MESSAGE

Environment

Displays a message in the status bar if it is enabled. This command is supported primarily for compatibility with dBASE IV. In *Visual* dBASE, you generally use the StatusMessage property of an object to display a message on the status bar when an object receives focus.

For more information about SET MESSAGE, see online Help.

SET MOUSE

Keyboard and mouse events

Use SET MOUSE to enable or disable the mouse . This command is supported primarily for compatibility with dBASE IV. In *Visual* dBASE applications, you would generally not disable mouse capabilities.

Syntax
SET MOUSE ON | off

Default
The default for SET MOUSE is ON.

Description
Use SET MOUSE OFF to remove the mouse pointer from the screen. Once the mouse pointer is removed, dBASE ignores all subsequent mouse actions. To enable the mouse again, use SET MOUSE ON.

See Also
ISMOUSE(), MCOL(), MROW(), ON MOUSE

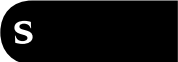
SET NEAR

Table organization

Specifies where to move the record pointer after a FIND, SEEK, or SEEK() operation fails to find an exact match.

Syntax
SET NEAR on | OFF

Default
The default for SET NEAR is OFF. To change the default, update the NEAR setting in DBASEWIN.INI. To do so, either use the SET command to specify the setting interactively, or enter the NEAR parameter directly in DBASEWIN.INI.



Description

Use SET NEAR to position the record pointer in an indexed table close to a particular key value when a search does not find an exact match. When SET NEAR is ON, the record pointer is set to the record closest to the key expression searched for but not found with FIND, SEEK, or SEEK(). When SET NEAR is OFF and a search is unsuccessful, the record pointer is positioned at the end of the file.

If you unsuccessfully search for a character, date, numeric, or float value when SET NEAR is ON, and the index is in ascending order, *Visual* dBASE positions the record pointer at the record whose key value follows the value searched for. When SET DELETED is ON or a filter is set with the SET FILTER command, SET NEAR disregards deleted or filtered records in determining the record nearest the key value expression.

With SET NEAR ON, FOUND() returns .T. for an exact match or .F. for a near match. With SET NEAR OFF, FOUND() returns .F. if no match occurs.

Example

The following example uses SET NEAR to control the placement of the record pointer if no matching record is found in an indexed table:

```
USE Company EXCLUSIVE
INDEX ON Zip_P_code TAG Zip
SET NEAR ON
SEEK "55555"
? EOF(), Zip_P_code
SET NEAR OFF
SEEK "55555"
? EOF(), Zip_P_code
```

There is no Zip_P_code 55555 in the table. With NEAR OFF, the record pointer is positioned at EOF(). With NEAR ON, it is positioned at the nearest record, deleted or not.

Portability

Not supported in dBASE III PLUS.

See Also

EOF(), FIND, FOUND(), LOCATE, SEEK, SEEK(), SET(), SET DELETED, SET FILTER

SET ODOMETER

Environment

Specifies how frequently dBASE updates and displays record counter information in the status bar if it is enabled. This command is supported primarily for compatibility with dBASE IV. *Visual* dBASE displays a progress meter while certain commands are being carried out.

For complete syntax information on SET ODOMETER, see online Help.

SET ORDER

Table organization

Specifies an open index file or tag as the master index of a table.

Syntax

SET ORDER TO [*<index position expN>*]

or

SET ORDER TO [*<.ndx filename 1>*]
[NOSAVE]

or

SET ORDER TO
[TAG] *<tag name>*
[OF *<filename 2>* | ?]
[NOSAVE]

<index position expN> A number specifying the position of an index in the list of open .NDX files. This option is provided for compatibility with dBASE III PLUS and cannot be used if there are any .MDX files open. If *<index position expN>* evaluates to 0, the table appears unindexed, in record number or *natural* order.

<.ndx filename 1> Specifies the name of an .NDX file created on a dBASE table.

[TAG] <tag name> Specifies the name of an index tag open in an .MDX file (or a single or multiple field index created on a Paradox or SQL table). The TAG keyword is included for readability only; TAG has no affect on the operation of the command. For Paradox tables, if you specify SET ORDER TO without including an index tag name, the primary index is used as the master index, if it exists.

OF <filename 2> | ? Specifies the multiple index file that contains the index tag you want to control the order of the current table. OF ? displays a dialog box, in which you select an index file. If you specify a file without including its path, *Visual* dBASE looks for the file in the current directory, then in the path you specify with SET PATH. If you specify a file without including its extension, *Visual* dBASE assumes an .MDX extension.

If you use the [TAG] *<.ndx filename 1>* | *<tag name>* option but don't specify the name of a .MDX index file, *Visual* dBASE searches first for an .NDX index file, and then searches for the tag name in the production .MDX file.

NOSAVE Used to delete a temporary index after the associated table is closed. If you decide after choosing this option that you want to keep the index, open the index again using SET ORDER without the NOSAVE option, before you close the table.

Description

Use SET ORDER to change the master index of a table without having to close and reopen indexes. You can choose the master index from the list of .NDX files or .MDX index tags opened with the SET INDEX or USE...INDEX commands.

If you specify the order with *<index position expN>*, use the index order defined with the SET INDEX command. You can use this option only if no .MDX files are open. If you

specify SET ORDER without specifying an index, or if *<index position expN>* evaluates to 0, the table appears unindexed, in record number order.

When accessing a Paradox table, specifying SET ORDER TO without an argument resets the Paradox table to the primary index record order, if a primary index exists.

Example

The following example uses SET ORDER to specify which index file or tag is the active index:

```
USE Company EXCLUSIVE
INDEX ON CompCode TAG CompCode
INDEX ON Company TAG Company
INDEX ON City TAG City
SET INDEX TO CompCode, Company, City
BROWSE FIELDS CompCode, Company, City ;
    TITLE "Compcode order"
SET ORDER TO City
BROWSE FIELDS CompCode, Company, City ;
    TITLE "City order"
SET ORDER TO Company
BROWSE FIELDS CompCode, Company, City ;
    TITLE "Company order"
SET ORDER TO                                && return to natural order
```

See Also

CLOSE..., INDEX, KEY(), MDX(), NDX(), ORDER(), REINDEX, SET INDEX, TAG(), TAGCOUNT(), TAGNO(), USE

SET PATH

Disk and file utilities

Specifies the directory search route that dBASE follows to find files that are not in the current directory.

Syntax

```
SET PATH TO
[<path list>]
```

<path list> A list of (optional) drives and directories indicating the *search path*—one or more drives and directories you want dBASE to search for files. Separate each directory path name with commas, semicolons, or spaces. The syntax for this command is similar to the PATH command available in DOS. Unlike DOS, however, you can use spaces or commas as well as semicolons to separate paths on the list; DOS accepts only semicolons.

Default

The default for SET PATH is empty (no path). To change the default, update the PATH setting in DBASEWIN.INI with as many valid drives and directories as you want. To do so, either use the SET command to specify the setting interactively, or enter the PATH parameter directly in DBASEWIN.INI.

Description

Use SET PATH to establish a search path to access files located on directories other than the current directory. When no SET PATH setting exists and you don't provide the full path name when you specify a file name, dBASE searches for that file only in the current directory.

The order in which you list drives and directories with SET PATH TO *<path list>* is the order dBASE searches for a file in that search path. Use SET PATH when an application's files are in several directories.

A path name without a drive letter or beginning backslash begins at the current directory on the current drive. Use two periods (..) to signify the parent directory of the current directory, and use a beginning backslash to signify the root directory of the current drive.

SET PATH TO without the option *<path list>* resets the search path to the default value (no path).

Example

The following example uses SET PATH, opens a table in the path and uses SET FULLPATH to show the subdirectory used:

```
* Current directory is d:\example
SET PATH TO c:\temp prg, d:\example\pdx
* 'prg' refers to d:\example\prg
USE Lineitems
SET FULLPATH ON
? DBF() && D:\EXAMPLE\PDOX\LINEITEMS.DBF
SET FULLPATH OFF
```

See Also

DISPLAY STATUS, LIST STATUS, SET DIRECTORY

SET PCOL

Printing

Sets the printing column position of a printer, which is the value of PCOL().

Syntax

SET PCOL TO *<expN>*

<expN> The column number to which to set PCOL(). The valid range is 0 to 32,767, inclusive.

S

Description

Use SET PCOL to set the horizontal printing position of a printer, which is the value the PCOL() function returns. Subsequent @...SAY commands that specify a column position on the current line print in a column position relative to the new PCOL() value. Generally, you use the command SET PCOL TO 0 to reset the printer column to the left edge of the page.

SET PCOL doesn't affect the printing of @...SAY commands that use relative addressing with PCOL(). For example, the following command line prints "Hello" beginning 10 spaces to the right of the current column, regardless of the value of PCOL().

```
@ 1,PCOL() + 10 SAY "Hello"
```

When you move the printing position to a new line, dBASE reinitializes PCOL() to 0, so SET PCOL affects the value of PCOL() for the current line only. When you send output to your printer, dBASE updates PCOL() by adding 1 to the current PCOL() value for each character it sends to the printer. The printing position moves 1 column for each character the printer prints.

When you send a printer control code or escape sequence to your printer, the printing position doesn't move. (Printer control codes and escape sequences are strings that give the printer instructions, such as to print underlining, boldface type, or different fonts.) Although control codes and escape sequences don't move the printing position, dBASE nonetheless increments the PCOL() value by the number of characters that you send to the printer. Each control code character increments the value of PCOL() by 1 just like any other character. As a result, the value of PCOL() might not reflect the actual printing position. Use SET PCOL to reset the value of PCOL() to the same value as the printing position.

To send a control code to the printer without changing the value of PCOL(), save the current value of PCOL() to a memory variable, send the control code to the printer, then SET PCOL to the contents of the memory variable.

Example

The following example writes "Jack & Jill" to the printer. It uses PCOL() to note the column position three times, at the beginning, after "Jack", and after "Jill":

```
SET TALK OFF
SET PRINTER ON
* now ?s are directed to printer
?                                && printer at col 0 of next line
beginpos=pcol()                 && note the current column
?? "Jack"
lastjackpos=pcol()
?? " & Jill"
lastjillpos=pcol()
* prints:
* Jack & Jill
SET PRINTER OFF
CLOSE PRINTER
? beginpos                      && 0.00
? lastjackpos                   && 4.00
? lastjillpos                   && 11.00
* Displays column positions for reference
```

Portability

Not supported in dBASE III PLUS or dBASE IV.

See Also

@...SAY...GET, PCOL(), PROW(), SET PROW

SET POINT

Numeric data

Specifies the character that separates decimal digits from integer digits in numeric display.

Syntax

SET POINT TO

[<expC>]

<expC> The character representing the decimal point. You can specify more than one character, but dBASE uses only the first one. If you specify a number as a character for <expC> (for example, "3"), dBASE returns an error.

Default

The default for SET POINT is set by the International option of the Windows Control Panel. To change the default, set the POINT parameter in DBASEWIN.INI. To do so, either use the SET command to specify the setting interactively, or enter the POINT parameter directly in DBASEWIN.INI.

Description

SET POINT affects both numeric input and display with commands such as EDIT. SET POINT also affects numeric display with commands such as DISPLAY MEMORY, STORE, =, and the PICTURE "." template character. You must use the period in the PICTURE option, regardless of the setting of SET POINT. See Picture in Chapter 8 for more information on the PICTURE option.

SET POINT affects only the display of numeric expressions in dBASE syntax, not their *input*. Only a period is valid as a decimal point in numeric input. For example, if you SET POINT TO "," (comma) and issue the following command, dBASE returns an error:

```
? MAX(123,4, 123,5)
```

The correct syntax is

```
? MAX(123.4, 123.5)
```

SET POINT TO without the <expC> option resets the decimal character to the default set with the International option of the Windows Control Panel.

S

Example

The following example uses SET POINT to control the display of numeric data:

```
SET DECIMALS TO 2                && Default
SET CURRENCY LEFT                && Default
SET SEPARATOR TO "."
SET POINT TO ","
SET CURRENCY TO "FR"
? 23445.95 PICTURE "$999,999.99" && Returns FR23.445,95
```

SET PRECISION

```
? 2345.95 PICTURE "$999,999.99"      && Returns FFR2.345,95
? 345.95 PICTURE "$999,999.99"      && Returns FFFFR345,95
? 345.95 PICTURE "@$999,999.99"     && Returns FR345,95
```

Another example of SET POINT is included in the INT() example.

Portability

Not supported in dBASE III PLUS.

See Also

SET DECIMALS, SET SEPARATOR, STORE

SET PRECISION

Numeric data

Determines the number of digits dBASE uses when comparing numbers.

Syntax

```
SET PRECISION TO
[<expN>]
```

<expN> The number of digits, from 10 to 19.

Default

The default for SET PRECISION is 16. To change the default, set the PRECISION parameter in DBASEWIN.INI. To do so, either use the SET command to specify the setting interactively, or enter the PRECISION parameter directly in DBASEWIN.INI.

Description

Use SET PRECISION to change the accuracy, or precision, of numeric comparisons. You can set precision from 10 to 19 digits.

SET PRECISION affects data comparisons, but not mathematical computations or data display. In math computations, the precision is always 19 digits. To change the number of decimal places dBASE displays, use SET DECIMALS.

Example

The following example demonstrates how the precision setting affects data comparisons and mathematical computations:

```
SET DECIMALS TO 18      && Max value
SET PRECISION TO 19     && Max value
x = 0.12345678901234567
y = 0.12345678901234568
? x = y                  && Returns .F.
? x + y                  && Returns 0.246913578024691350

SET PRECISION TO 16
? x = y                  && Now returns .T.
? x + y                  && Still returns 0.246913578024691350
```



```
SET DECIMALS TO 8
```

```
? x + y
```

```
&& Returns 0.24691358
```

Portability

Not supported in dBASE III PLUS. In dBASE IV, SET PRECISION affects mathematical computations, but not comparisons of numbers.

See Also

SET DECIMALS

SET PRINTER

Printing

The SET PRINTER TO setting specifies a file to receive streaming output, or uses a device code recognized by the Windows Print Manager to designate a printer. The On/Off setting controls whether dBASE also directs streaming output that appears in the Command window to the device or file specified by SET PRINTER TO.

Syntax

SET PRINTER on | OFF

SET PRINTER TO

[[FILE] <filename> | ? | <filename skeleton>] | [<device>]

FILE Use this keyword if you want to bypass the Windows print driver so that no printer control codes are written to the file. If you omit the keyword, the Windows print driver output is written to the file.

<filename> | ? | <filename skeleton> The text file to send output to instead of the printer. By default, dBASE assigns a .PRT extension to <filename> and saves the file in the current directory. The ? and <filename skeleton> options display a dialog box, in which you specify the name of the target file and the directory to save it in.

<device> The printer port of the printer to send output to. Specify printers and their ports with the Windows Control Panel.

Default

The default for SET PRINTER is OFF. To change the default, set the PRINT parameter in the [OnOffCommandSettings] section in DBASEWIN.INI. The default for SET PRINTER TO is the default printer you specify with the Windows Control Panel.

Description

Use SET PRINTER TO to direct *streaming output* from commands such as ?, ??, and LIST to a printer or a text file. SET PRINTER TO with no option sends this output to the default printer. For more information about streaming and non-streaming output, see Chapter 24 in the *Programmer's Guide*.

Use SET PRINTER ON/OFF to enable or disable the printer you specify with SET PRINTER TO.

SET PRINTER

To send streaming output to a file rather than the printer, issue SET PRINTER TO FILE *<filename>*. When you issue SET PRINTER TO FILE *<filename>*, issuing SET PRINTER ON directs streaming output to the text file *<filename>* rather than to the printer. The file has the default extension of .PRT.

When SET PRINTER is OFF, dBASE directs streaming output only to the result pane of the Command window. SET PRINTER must be ON to output data to a text file unless you issue a command with its TO PRINTER option. The following example illustrates this behavior:

```
SET PRINTER OFF
SET PRINTER TO FILE test.prt
TYPE file.txt                && displays on screen only
TYPE file.txt TO PRINT       && output sent to screen and test.prt
```

To send non-streaming output to the printer, use SET DEVICE TO PRINTER. To send non-streaming output to a text file, use SET DEVICE TO FILE.

Example

The following example uses SET PRINTER ON and OFF:

```
SET PRINTER TO                && Set printer to default
SET PRINTER ON
? "Hello"                    && to printer
SET PRINTER OFF
? prow(),pcol()              && only displayed to screen
CLOSE PRINTER                && initiate printing
SET PRINTER TO               && resets to default
SET PRINTER TO PRN           && sets to DOS output device
SET PRINTER TO LPT1
SET PRINTER TO NUL
SET PRINTER TO FILE Test
* Test.prt receives streaming output, including any control codes,
* that would have gone to the printer.
```

Portability

In dBASE III PLUS, use SET PRINT.

See Also

SET ALTERNATE, SET CONSOLE, SET DEVICE

SET PROCEDURE

Programs

Opens a dBASE program file (.PRG), making all procedures and user-defined functions (UDFs) in the file available for execution.

Syntax

```
SET PROCEDURE TO
[<filename> | ? | <filename skeleton>]
[ADDITIVE]
```

<filename> | ? | <filename skeleton> The procedure file to open. The ? and <filename skeleton> options display a dialog box, from which you can select a file. If you specify a file without including its path, dBASE looks for the file in the current directory, then in the path you specify with SET PATH. If you specify a file without including its extension, dBASE assumes .PRO (a compiled object file). If dBASE can't find a .PRO file, it looks for a .PRG file (a source file). If dBASE finds a .PRG file, it compiles it.

ADDITIVE Opens the procedure file(s) without closing any you've opened with previous SET PROCEDURE statements. SET PROCEDURE TO <filename> (**without** the ADDITIVE option) closes all procedure files you've opened with previous SET PROCEDURE statements.

Description

To execute a procedure or UDF, dBASE must have access to the file containing it. When dBASE encounters a call to a procedure or UDF, it looks for the procedure or UDF in specific places in a specific order. One of the places dBASE looks is in a file opened with SET PROCEDURE. See the DO command for an explanation of the search path and order dBASE uses.

To make the procedures or UDFs in a program available to other programs, place the following statement in the program:

```
SET PROCEDURE TO PROGRAM(1) ADDITIVE
```

If you issue SET PROCEDURE TO with no options, dBASE closes all procedure files you've opened with SET PROCEDURE. If you want to close only specific procedure files, use CLOSE PROCEDURE. The maximum number of open procedure files depends on available memory.

For more information about working with procedure files, see Chapter 4 in the *Programmer's Guide*.

Example

The following program lines use SET PROCEDURE to set and add various procedure files so that all programs can access the procedures in the procedure files. Finally, all procedure files are released:

```
SET PROC TO SoundLib
* All programs can access all procedures in SoundLib.PRG
SET PROCEDURE TO PictLib ADDITIVE
* All programs can now access all procedures in SoundLib.PRG and PictLib.PRG
SET PROCEDURE TO PictLib
```

SET PROCEDURE

```
* Only PictLib can be accessed
SET PROCEDURE TO PROGRAM(1) ADDITIVE
* The current file, whatever its name, is now also a procedure file
SET PROCEDURE TO
* All procedure files are released and no longer available
```

The following example creates an entry form with three RadioButtons to select the desired conversion factor. SET PROCEDURE TO Cnvrt.PRG makes the function Metric in the procedure file Cnvrt available when the user clicks on entry field Answ:

```
** Metric Conversion Program **
SET PROCEDURE TO Cnvrt.PRG
f=NEW Convert()
f.OPEN()
CLASS Convert OF FORM
  this.Top=2
  this.Left=2
  this.Width=50
  this.Height=18
  this.Text= "Conversion Utility"
  DEFINE ENTRYFIELD Amt OF THIS AT 4,15;
    PROPERTY Value 0, Width 8
  DEFINE TEXT Ln1 OF THIS AT 2,6;
    PROPERTY;
    Text "Enter Amount; Select a RadioButton",;
    Width 45
  DEFINE RadioButton Inches OF THIS AT 6,8;
    PROPERTY Text "Inches to Centimeters",;
    Width 22, Value .F.
  DEFINE RadioButton Pounds OF THIS AT 8,8;
    PROPERTY Text "Pounds to Kilograms",;
    Width 21, Value .F.
  DEFINE RadioButton Degrees OF THIS AT 10,8;
    PROPERTY Text "Degrees F to C",;
    Width 21, Value .F.
  DEFINE TEXT Ln2 OF THIS AT 12,8;
    PROPERTY Text "Results:",;
    Width 15
  DEFINE ENTRYFIELD Answ OF THIS AT 12,19;
    PROPERTY Value 0, Width 17,,
    OnGotFocus Metric
  DEFINE PUSHBUTTON Exit OF THIS AT 15,15;
    PROPERTY TEXT "Exit", OnClick {;Form.Close()}
ENDCLASS
```

The following conversion function resides in a .PRG file named Cnvrt.PRG and is made available to the main program by the command line SET PROCEDURE TO Cnvrt.PRG

```
***Cnvrt.PRG***
FUNCTION Metric
DO CASE
  CASE Form.Inches.Value
    Form.Answ.Value = LTRIM(STR(Form.Amt.Value;
      * 2.54,10,2))+ " Centimeters"
  CASE Form.Pounds.Value
    Form.Answ.Value = LTRIM(STR(Form.Amt.Value;
```

```

    * .454,10,2))+" Kilograms"
CASE Form.Degrees.Value
  Form.Answ.Value = LTRIM(STR((Form.Amt.Value;
    -32)* (5/9),10,2))+" Degrees C"
ENDCASE
RETURN .T.

```

Portability

The `?`, `<filename skeleton>`, and `ADDITIVE` options aren't supported in dBASE III PLUS or dBASE IV. In dBASE IV, a file opened with `SET PROCEDURE` is always searched before a file opened with `SET LIBRARY`. In *Visual* dBASE, the first file opened is the first one searched.

See Also

`CLOSE...`, `COMPILE`, `DO`, `FUNCTION`, `PARAMETERS`, `PROCEDURE`, `RETURN`, `SET()`, `SET LIBRARY`

SET PROW

Printing

Sets the current row position of a printer's print head, which is the value of `PROW()`.

Syntax

`SET PROW TO <expN>`

<expN> The row number to which to set `PROW()`. The valid range is 0 to 32,767, inclusive.

Description

Use `SET PROW` to set the vertical printing position of a printer, which is the value the `PROW()` function returns. Subsequent `@...SAY` commands that specify a row position print relative to the new `PROW()` value. Generally, you use the command `SET PROW TO 0` to reset the printer row to top-of-page.

`SET PROW` doesn't affect the printing of `@...SAY` commands that use relative addressing with `PROW()`. For example, the following command line prints the word "Hello" two rows below the current row, regardless of the value of `PROW()`.

```
@ PROW() + 2,0 SAY "Hello"
```

Example

The following example uses `SET PROW` to print relative to the current row:

```

SET DEVICE TO PRINTER
@ 10,0 say "Tenth row"
SET PROW TO 0
* reset prow, previous row 10 is now row 0
@ 2,0 say "2nd row relative to new setting"
SET DEVICE TO SCREEN
CLOSE PRINTER

```

Portability

Not supported in dBASE III PLUS or dBASE IV.

See Also

@...SAY...GET, PCOL(), PROW(), SET PCOL

SET REFRESH

Shared data

Determines how often dBASE refreshes the workstation screen with table information from the server.

Syntax

SET REFRESH TO <expN>

<expN> A time interval expressed in seconds from 0 to 3,600 (1 hour), inclusive.

Default

The default for SET REFRESH is 0, meaning dBASE doesn't update the screen. To change the default, set the REFRESH parameter in DBASEWIN.INI. To do so, either use the SET command to specify the setting interactively, or enter the REFRESH parameter directly in DBASEWIN.INI.

Description

Use SET REFRESH to set a refresh interval when working with shared tables on a network. Then, when you use BROWSE, EDIT, or CHANGE to edit shared tables, your screen refreshes at the interval you set, showing you changes made by other users on the network to the same tables.

If another user has a lock on the file or records you're currently viewing, the file or records won't be refreshed until that user releases the lock.

Example

```
USE Employee
CONVERT
SET REFRESH TO 5
BROWSE
```

See Also

BROWSE, CHANGE, EDIT, FLOCK(), RLOCK()

SET RELATION

Table organization

Links two or more open tables with common fields or expressions.

Syntax

```
SET RELATION TO
[<key exp list 1> | <expN 1>
  INTO <child table alias 1>
  [, <key exp list 2> | <expN 2>
    INTO <child table alias 2>] ... ]
[ADDITIVE]]
```

For dBASE tables, you can also specify options to restrict processing of unlinked child table records and specify additional data integrity rules:

```
SET RELATION TO
[<key exp list 1> | <expN 1>
  INTO <child table alias 1>
  [CONSTRAIN
  [INTEGRITY
    [CASCADE | RESTRICTED]]]
  [, <key exp list 2> | <expN 2>
    INTO <child table alias 2>]
  [CONSTRAIN
  [INTEGRITY
    [CASCADE | RESTRICTED]]]...]
[ADDITIVE]
```

<key exp list 1> The key expression or field list that is common to both the current table and a child table and links both tables. When specifying the INTEGRITY option, you can specify a key field but not an expression. The child table must be indexed on the key field and that index must be the master index in use for the child table. An index based on the same key field must also be defined for the parent table but it doesn't need to be the master index for that table.

<expN 1> INTO <child table alias> For dBASE tables only, you can specify <expN> to link records in a child table. When <expN> is RECNO(), Visual dBASE links the current table to a child table by corresponding record numbers, in which case, the child table doesn't have to be indexed.

INTO <child table alias> <alias> specifies the child table linked to the current table. You can specify a work area number (1 through 225), letter (A through J), or alias name. The work area letter or alias name must be enclosed in quotes.

<key exp list 2> | <expN 2> INTO <alias 2> ...] Specifies additional relationships from the current table into other tables.

CONSTRAIN Limits records processed in the child table to those matching the key expression in the parent table.

INTEGRITY [CASCADE | RESTRICTED] Specifies rules that control the addition or deletion of records in the child table. If INTEGRITY is specified, key fields of records added to child tables via APPEND, APPEND BLANK, BROWSE, EDIT, or INSERT commands are set to values matching those of the parent table. Also, when you delete records or change

the value of key fields in the parent table, a dialog box appears, in which you can choose to delete all related child records (performing a cascade delete).

The CASCADE and RESTRICTED options specify rules to follow when adding or deleting records in a program (so the dialog box does not appear). The CASCADE option specifies that all child table records with matching key field values are deleted when you delete a record or change the key value of a record in the parent table. The RESTRICTED option prevents deletions or changes to records in the parent table if the child table contains records with matching key field values.

ADDITIVE Adds the new relation to any existing ones. Without ADDITIVE, SET RELATION clears existing relations before establishing the new relation.

Description

Use SET RELATION to establish a link between open tables based on common fields or expressions. Remember that you can also define relations and link tables automatically by using the CREATE | MODIFY QUERY or VIEW commands, which let you save relations to a .QBE file.

Before setting a relation, open each table in a separate work area. When a relation is set, the table in the current work area is referred to as the *parent* table, and a table linked to the parent table by the specified key is called a *child* table. The child table must be indexed on the fields or expressions that link tables and that index must be the master index in use for the child table.

A relation between tables is usually set through common keys specified by *<key exp list>*. The relating expression can be any expression derived from the parent table that matches the keys of the child table master index. The keys may be a single field or a set of concatenated fields contained in each table. The fields in each table can have different names but must contain the same type of data. For Paradox and SQL tables, you can specify single or composite index key fields.

For dBASE tables only, if you specify a numeric expression to link tables, the parent table is always linked to record numbers in a child table specified by the numeric expression (typically, the RECNO() function). This causes record 1 in the parent table to be linked to record 1 in the child table, record 2 in the parent table linked to record 2 in the child table, and so on.

SET RELATION clears existing relations before establishing a new one, unless you use the ADDITIVE option. SET RELATION TO without any arguments clears existing relations from the current table without establishing any new relations.

For dBASE tables, the CONSTRAIN and INTEGRITY options control processing of linked records. Using these options requires that the parent table not contain any records with duplicate key values. If a parent table has duplicate records, copy the table to a temporary table, index the temporary table using the UNIQUE option, and then use COPY to copy the table back to the original table name. After setting up a relation with either CONSTRAIN or INTEGRITY, the uniqueness of key values in the parent table is strictly enforced.

The CONSTRAIN option restricts access in the child table to only those records whose key values match records in a parent table. This is the same as using SET KEY TO on the key field of the child table. As a result, you can't use SET KEY TO and CONSTRAIN at

the same time. If a SET KEY TO operation is in effect on the child table when you specify CONSTRAIN with SET RELATION, *Visual* dBASE returns a "SET KEY in use in alias" message. If the CONSTRAIN option is in effect when SET KEY TO is specified, *Visual* dBASE returns the error "CONSTRAIN is active." You can use SET FILTER with the CONSTRAIN option, if you want to specify additional conditions to qualify records in a child table.

The INTEGRITY option specifies rules that control the addition or deletion of records in the child table. If INTEGRITY is specified, when you add records using APPEND, APPEND BLANK, BROWSE, EDIT, or INSERT commands, the key fields of records added to child tables are set to values matching those of the parent table. If you add new records to the parent table, values in the key field must be unique; otherwise, *Visual* dBASE returns the error "Key already exists." When you edit records in the child table, key fields in the child table are read-only.

Integrity rules assume SET DELETED is ON. Setting DELETED OFF is not recommended when using the INTEGRITY option since it could show linked records already marked for deletion.

If you specify the INTEGRITY option without the CASCADE or RESTRICTED keywords, when you delete records or change the value of key fields in the parent table, a dialog box appears, in which you can choose to delete all related child records (performing a cascade delete).

If you specify INTEGRITY with the CASCADE option, *Visual* dBASE automatically deletes all records in the child table that match the key value of records that are deleted or changed in the parent table. If a key value of a record in the parent table is changed to a value that already exists, *Visual* dBASE returns the error "Key already exists."

If you specify INTEGRITY with the RESTRICTED option, *Visual* dBASE prevents you from deleting records in the parent table with linked child table records. If you attempt to delete records or change the key value of records in the parent table while records with matching key values still exist in the child table, *Visual* dBASE returns the error "Linked records still exist in alias" and the parent table record is not deleted or changed. You can delete records in a child table unless that table is itself a parent and an integrity rule is defined that restricts deletions to its child table records.

More than one relation can be defined from the same table. Also, more than one relation can be set from the same parent table if you use the ADDITIVE option or if you specify multiple relations with the same SET RELATION command. You can also establish additional relations from a child table, thus defining a chain of relations. Cyclic relations aren't allowed; that is, *Visual* dBASE returns an error if you attempt to define a relation from a child table back into its parent table.

When a relation is set from a parent table to a child table, the relation can be accessed only from the work area that contains the parent table. To access fields of the child table from the current work area, use the alias pointer (->) and prefix the name of fields in the child table by its alias name.

If a matching record can't be found in a linked table, the linked table is positioned at the end-of-file, and EOF() returns .T. The setting of SET NEAR does not affect positioning of the record pointer in child tables. When the INTEGRITY option is used, if the parent

table is positioned at the end of file, *Visual* dBASE returns the error "No matching parent record."

When a SET SKIP list is active, the record pointer is advanced in each table, starting with the last work area in the relation chain and moving up the chain toward the parent table.

Example

The following example uses SET RELATION to create a relationship based on the key field, CompCode, between the parent table, Company, and the child table, Contact:

```
CLOSE ALL
USE Contact EXCLUSIVE
INDEX ON Compcode TAG Compcode
SELECT 2
USE Company EXCLUSIVE
INDEX ON Company TAG Company
SET RELATION TO CompCode INTO Contact
```

Now the contact and the company for which they are a contact are connected. Contact, the child table, must be indexed on Compcode. Company does not need to be indexed on Compcode.

If you want to add a second relationship for Company, use the ADDITIVE clause. In this example, a relationship to the TypeCo table is added. TypeCo is a code table that contains the TYPE field used in the company table (for example, "ISV") and a description in the TYPE_DESC field (for example, "Independent Software Vendor"):

```
SELECT SELECT()                && Select the next open work area
USE TypeCo EXCLUSIVE           && Contains a description of the type of company
INDEX ON TYPE TAG TYPE
SELECT Company
SET RELATION TO Type INTO TypeCo ADDITIVE
```

Now Company is simultaneously related to Contact via field Compcode and to TypeCo via the Type field.

```
DISPLAY ALL Company,Contact->Contact,;
      TypeCo->Type_desc
* All three tables are accessed
```

See Also

CREATE VIEW FROM ENVIRONMENT, JOIN, SELECT, SET FIELDS, SET FILTER, SET SKIP, SET VIEW, SKIP

SET REPROCESS

Shared data

Specifies the number of times dBASE tries to lock a file or one or more records before generating an error or returning .F.

Syntax

SET REPROCESS TO <expN>

<expN> A number from -1 to 32,000, inclusive, that is the number of times for dBASE to try to lock a file or one or more records.

Default

The default for SET REPROCESS is 0, meaning dBASE doesn't try to lock a file or one or more records after the original attempt. To change the default, set the REPROCESS parameter in DBASEWIN.INI. To do so, either use the SET command to specify the setting interactively, or enter the REPROCESS parameter directly in DBASEWIN.INI.

Description

Use SET REPROCESS to specify how many times dBASE should try to lock a file or one or more records before generating an error or returning .F. SET REPROCESS affects RLOCK(), LOCK(), and FLOCK(), and all commands and functions that automatically attempt to lock a file or records.

SET REPROCESS TO 0 causes dBASE to prompt you to retry setting a lock or to cancel. If you cancel, RLOCK(), LOCK(), and FLOCK() return .F., and a command that automatically attempts to lock a file or record returns an error.

Setting SET REPROCESS to a number greater than 0 causes dBASE to retry setting a lock the specified number of times without prompting you to confirm each retry or to cancel.

SET REPROCESS TO -1 causes dBASE to retry setting a lock without prompting you to retry or cancel, and to continue retrying until the lock succeeds.

Example

See FLOCK() for an example for SET REPROCESS.

Portability

Not supported in dBASE III PLUS.

See Also

FLOCK(), ON ERROR, ON NETERROR, RETRY, RLOCK(), SET LOCK

S

SET SAFETY

Environment

Determines whether dBASE asks for confirmation before overwriting a file or removing records from a table when you issue ZAP.

Syntax

SET SAFETY ON | off

Default

The default for SET SAFETY is ON. To change the default, set the SAFETY parameter in DBASEWIN.INI. To do so, either use the SET command to specify the setting interactively, or enter the SAFETY parameter directly in DBASEWIN.INI.

Description

When SET SAFETY is ON, dBASE prompts for confirmation before overwriting a file or removing records from a table when you issue ZAP. If you want your program to control the interaction between dBASE and the user with regard to overwriting files, issue SET SAFETY OFF in your program.

SET SAFETY affects the following commands:

- Commands using a TO FILE option
- COPY
- COPY FILE
- COPY TO...STRUCTURE EXTENDED
- CREATE/MODIFY commands
- INDEX
- JOIN
- SAVE
- SET ALTERNATE TO
- SORT
- TOTAL
- UPDATE
- ZAP

Note SET TALK OFF does not suppress SET SAFETY warnings.

Example

This example zaps a table when safety is on and when it is off:

```
USE COMPANY
COPY TO TEMP
* Make a temporary table
USE TEMP EXCLUSIVE
SET SAFETY ON
ZAP
* A window titled ZAP appears and a message
* "Remove all records from TEMP.DBF" is shown.
* The user will have to OK zapping the TEMP.DBF
```

```

SET SAFETY OFF
ZAP
* The table is zapped automatically
SET SAFETY ON
* Turn safety back on
USE
DELETE FILE TEMP.DBF
DELETE FILE TEMP.DBT
* Delete the temporary table

```

See Also

SET TALK

SET SEPARATOR

Numeric data

Specifies the character that separates each group of three digits (whole numbers) to the left of the decimal point in the display of numbers greater than or equal to 1000.

Syntax

```

SET SEPARATOR TO
[<expC>]

```

<expC> The *whole-number separator*, which is the character that separates each group of three digits to the left of the decimal point in the display of numbers greater than or equal to 1000. You can specify more than one character, but dBASE uses only the first one. If you specify a number as a character for <expC> (for example, "3"), dBASE returns an error.

Default

The default for SET SEPARATOR is set by the International option of the Windows Control Panel. To change the default, set the SEPARATOR parameter in DBASEWIN.INI. To do so, either use the SET command to specify the setting interactively, or enter the SEPARATOR parameter directly in DBASEWIN.INI.

Description

SET SEPARATOR affects only the PICTURE "," template character and the numeric display of byte totals for the commands DIR, DISPLAY FILES, and LIST FILES. For example, if you SET SEPARATOR TO "."(period) and issue the following, dBASE returns 123456 displayed as 123.456:

```
? 123456 PICTURE "999,999"
```

You must use the comma in the PICTURE function, regardless of the setting of SET SEPARATOR. For more information on the PICTURE option, see Picture in Chapter 8.

SET SEPARATOR TO without the <expC> option resets the separator to the default set with the International option of the Windows Control Panel.

Setting a whole-number separator with SET SEPARATOR doesn't affect the values of numbers, only their display.

Example

See SET POINT and INT() for examples of SET SEPARATOR.

Portability

Not supported in dBASE III PLUS.

See Also

SET POINT

SET SKIP

Table organization

Specifies how to advance the record pointer through records of linked tables.

Syntax

SET SKIP TO

[<alias 1> [, <alias 2>]...]

<alias1> [, <alias2>] ... The alias tables defined in a relation. You can specify a work area number (1 through 225), letter (A through J), or alias name. The work area letter or alias name must be enclosed in quotes. SET SKIP TO without any options cancels previous SET SKIP settings.

Description

SET SKIP works only with tables that have been linked with the SET RELATION command. Used together, the SET RELATION and SET SKIP commands determine the way in which the record pointer moves through parent and child tables.

Use SET SKIP when you set a relation from a parent table containing unique key values to child tables that contain duplicate key values, that is, a one-to-many relationship. SET SKIP causes commands that move the record pointer to move the pointer to every record with matching key values in a child table before moving the record pointer in the parent table. If you define a chain of relations and use SET SKIP to move from one table to the next down the chain, the record pointer moves to every record in the last child table before the pointer moves in its parent table.

Example

The following example uses SET RELATION to create a relationship based on the key field, CompCode, between the parent table, Company, and the child table, Contact. It then uses SET SKIP in two different ways. First it accesses all contacts for each company, and second it accesses only one contact per company:

```
CLOSE DATABASE
USE Contact EXCLUSIVE
INDEX ON Compcode+Contact TAG Compcont
* Compcode+Contact is used rather than just
* Compcode because now, for each company, the
* Contacts are in alphabetical order
SELECT 2
USE Company EXCLUSIVE
```

```

INDEX ON Company TAG Company
SET RELATION TO CompCode INTO Contact
* Now the tables and relationships are established
SET SKIP TO Contact
DISPLAY ALL Company,Contact->Contact
* All contacts for all companies are shown
SET SKIP TO
DISPLAY ALL Company,Contact->Contact
* Now only one contact per company is shown

```

Portability

Not supported in dBASE III PLUS.

See Also

SET RELATION, SKIP

SET SPACE

Input/Output

Determines whether dBASE inserts a space between expressions displayed or printed with a single ? or ?? command.

Syntax

SET SPACE ON | off

Default

The default for SET SPACE is ON. To change the default, set the SPACE parameter in DBASEWIN.INI.

Description

Use SET SPACE OFF when you use a single ? or ?? command to print a list of expressions and you don't want spaces between the expressions. If you want the expressions printed with spaces between them, issue SET SPACE ON.

SET SPACE has no effect on multiple ? or ?? commands. For example, if you issue the command ?? <exp> twice, the second instance of <exp> will be printed adjacent to the first, even if SET SPACE is ON. However, if SET SPACE is ON and you issue ?? <exp>, <exp> as a single command, there will be a space between the two instances of <exp>.

Example

This example displays a first and a last name using SET SPACE ON and then SET SPACE OFF:

```

Firstname="Rachel"
Lastname ="Jayes"
SET SPACE ON                && the default
? Firstname,Lastname
* Rachel Jayes
SET SPACE OFF
? Firstname,Lastname

```

S

SET STEP

- * RachelJayes
- * The two variables are not separated

Portability

Not supported in dBASE III PLUS, which always adds a space between printed expressions (as if SET SPACE were ON).

See Also

?, ??, LTRIM(), RTRIM(), TRIM()

SET STEP

Error handling and debugging

SET STEP ON opens the dBASE Debugger. This command is supported primarily for compatibility with dBASE IV. In *Visual* dBASE, use DEBUG to open the debugger.

For complete syntax information on SET STEP, see online Help.

SET TALK

Environment

Determines whether dBASE displays messages in the status bar, or displays memory variable assignments in the results pane of the Command window.

Syntax

SET TALK ON | off

Default

The default for SET TALK is ON. To change the default, set the TALK parameter in DBASEWIN.INI. To do so, either use the SET command to specify the setting interactively, or enter the TALK parameter directly in DBASEWIN.INI.

Description

When SET TALK is ON, dBASE uses the current SET ODOMETER setting to indicate when operations such as COUNT and SORT are in progress in the status bar. It also displays the results of memory variable assignments (using STORE or =) in the results pane of the Command window.

Depending on the amount of memory your system has and the amount of memory particular operations require, issuing SET TALK OFF might improve the performance of some operations.

Use SET TALK with SET ALTERNATE or SET DEVICE to send SET TALK output to a file or printer rather than to the results pane of the Command window.

Example

This example shows the effect of SET TALK ON and SET TALK OFF while creating a memory variable:


```

Oldtalk=SET("TALK")
SET TALK ON
First="Susan"                && Susan
Last="O'Shenko"             && O'Shenko
Name=Last+", "+First
SET TALK OFF
First="Tom"
Last="Frost"
Name=Last+", "+First
? Name
* Name will be set to "Frost, Tom" but this will not
* be displayed in the status bar
SET TALK &Oldtalk

```

When TALK is OFF, the assignment of "Tom" to First and "Frost" to Last and "Frost, Tom" to Name are not displayed.

In the following example Count displays to the status bar when TALK is ON:

```

CLOSE ALL
USE COMPANY
SET TALK ON                && Talk on
COUNT TO Recs             && Count displays in status bar
SET TALK OFF               && Talk off
COUNT TO Recs             && No display

```

See Also

SET ALTERNATE, SET CONSOLE, SET DEVICE, SET ODOMETER, STORE

SET TIME

Date and time data

Sets the system time.

Syntax

SET TIME TO <expC>

<expC> The time, which you must specify in one of the following formats:

- HH
- HH:MM or HH.MM
- HH:MM:SS or HH.MM.SS

Default

The default for the value of the system time is set by the Date/Time option of the Windows Control Panel.

Description

Use SET TIME to reset your system's clock. Subsequent values of TIME() and the time stamp of any files you save reflect the new time.

S

Example

See SET DATE TO for an example SET TIME.

Portability

Not supported in dBASE IV or dBASE III PLUS.

See Also

SET DATE TO, TIME()

SET TITLE

Table basics

Turns the catalog file title prompt on or off.

Syntax

SET TITLE ON | off

Default

The default for SET TITLE is ON. To change the default, set the TITLE parameter in DBASEWIN.INI. To change the default, update the TITLE setting in DBASEWIN.INI. To do so, either use the SET command to specify the setting interactively, or enter the TITLE parameter directly in DBASEWIN.INI.

Description

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 to enter a file title or description to accompany the file entry in the catalog. Otherwise, this prompt does not appear.

When you create files from the Command window and a catalog is open, files are added to the catalog, but you are not prompted to enter a description.

To enter a title or modify other information in a catalog file, use commands such as BROWSE, EDIT, or REPLACE. You can open and modify a catalog file in any work area, but you cannot modify a catalog opened by SET CATALOG.

Example

The following example uses SET TITLE OFF to disable the prompt to enter a file title for the catalog record when a newly created file is added to the open catalog:

```
SET CATALOG ON
SET CATALOG TO Learn
SET TITLE OFF
CREATE NewNames TYPE DBASE FROM Customer TYPE PARADOX
USE NewNames TYPE DBASE
BROWSE
```

See Also

CATALOG(), CREATE CATALOG, SELECT(), SET CATALOG, USE

SET TOPIC

Windows programming

Specifies a Help topic to display initially.

Syntax

SET TOPIC TO [*<expC>*]

<expC> The help topic keyword. If you omit *<expC>*, dBASE displays the Help Contents topic by default.

Default

The default for SET TOPIC is an empty string. To change the default, update the TOPIC setting in DBASEWIN.INI. To do so, either use the SET command to specify the setting interactively, or enter the TOPIC parameter directly in DBASEWIN.INI.

Description

Use SET TOPIC to display a specified topic each time:

- The HELP command is executed.
- The user presses *F1* while the cursor is in the Command window.

You specify the topic with *<expC>*, the help topic keyword. If you specify the entire keyword, the Help system displays the topic immediately. If you specify only initial characters, the Help system displays the Search dialog box, a tool that lets users select topics from a list. The first topic whose Help keyword has *<expC>* in its leftmost position is highlighted automatically. For example, executing SET TOPIC TO "CA", then HELP displays the Search dialog box with CALCULATE highlighted.

When the user presses *F1* while the cursor isn't in the Command window, the Help system is context-sensitive and ignores the SET TOPIC TO setting. For example, if you execute SET TOPIC TO "CA", then execute BROWSE, pressing *F1* displays information on the Table Editor rather than CALCULATE.

Example

The following example would change the help topic based on a menu choice:

```
DO CASE
CASE MenuChoice = 1
    SET TOPIC TO "BROWSE"
    BROWSE                                && dBASE BROWSE
CASE MenuChoice = 2
    SET TOPIC TO "EDIT"
    EDIT                                  && dBASE EDIT
CASE MenuChoice = 3
    SET TOPIC TO "SET"
    SetOptions                            && User procedure to set various settings. ;
                                         Topic will bring up search window in help with ;
                                         SET... topics displayed
CASE MenuChoice = 4
    QUIT
ENDCASE
```



Portability

Not supported in dBASE IV or dBASE III PLUS.

See Also

HELP, HelpFile, HelpID, SET HELP TO

SET TYPEAHEAD

Keyboard and mouse events

Sets the size of the typeahead buffer, where keystrokes are stored while dBASE is busy processing other data. This command is supported primarily for compatibility with dBASE IV. In *Visual* dBASE, forms do not use the typeahead buffer.

For complete syntax information on SET TYPEAHEAD, see online Help. For information about working with forms, see the Forms chapters in the *User's Guide*.

SET UNIQUE

Table organization

Determines if records with duplicate key values appear in an index file.

Syntax

SET UNIQUE on | OFF

Default

The default for SET UNIQUE is OFF. To change the default, update the UNIQUE setting in DBASEWIN.INI. To do so, either use the SET command to specify the setting interactively, or enter the UNIQUE parameter directly in DBASEWIN.INI.

Description

Use SET UNIQUE ON to include only the first record with the same key value in .MDX and .NDX indexes created with the INDEX command. When SET UNIQUE is OFF, indexes you create can include records with identical key values; records with identical keys are arranged by record number in the index.. Whenever you reindex an index file, *Visual* dBASE maintains the index in the same way it was created.

Visual dBASE processes unique indexes only once. Therefore, a previously hidden key value is not automatically updated when it is changed. Also, if you append a record that contains an index key that is already in the index file, the new record is not added to the index file, although the table is updated with the new record. REINDEX explicitly updates all key values in a unique index.

Following SET UNIQUE ON with an INDEX command is equivalent to issuing the single command INDEX...UNIQUE.

Example

The following example uses SET UNIQUE to make an index with just one record per state:

```

USE Clients EXCLUSIVE
INDEX ON State_Prov TAG State
SET UNIQUE ON
INDEX ON State_Prov TAG UnqState
SET UNIQUE OFF && RESET UNIQUE
SET ORDER TO TAG State
BROWSE FIELDS State_Prov, Company ;
    TITLE "All records"
SET ORDER TO TAG Unqstate
BROWSE FIELDS State_Prov, Company ;
    TITLE "One record per State"
COUNT TO numstates
WAIT "There are companies in " + LTRIM(STR(numstates)) + " states"

```

See Also

INDEX, REINDEX, SET(), SET INDEX, SET ORDER, UNIQUE(), USE

SET VIEW

Table organization

Opens a previously defined query or view file.

Syntax

SET VIEW TO <filename> | ? | <filename skeleton>

<filename> | ? | <filename skeleton> The query or view file containing the settings to define the current working environment or view. SET VIEW TO ? and SET VIEW TO <filename skeleton> display a dialog box, in which you select a view file. If you specify a file without including its path, *Visual* dBASE looks for the file in the current directory, then in the path you specify with SET PATH. If you specify a file without including its file-name extension, *Visual* dBASE looks for a .QBE, then a .VUE file.

Description

Use SET VIEW to change the working environment to one that was previously defined by CREATE QUERY, CREATE VIEW, or CREATE VIEW...FROM ENVIRONMENT. The working environment includes open tables and index files, all relations, the active fields list, and filter conditions.

Example

See CREATE VIEW ... FROM ENVIRONMENT for an example of SET VIEW.

See Also

CREATE QUERY, CREATE VIEW, CREATE VIEW...FROM ENVIRONMENT

SET WINDOW OF MEMO

Specifies a window to use while editing the contents of a memo field.

Syntax

SET WINDOW OF MEMO TO [*<window name>*]

TO *<window name>* Specifies the name of a previously defined window to be used for editing the contents of a memo field.

Description

Use the SET WINDOW OF MEMO command to use a previously defined window to edit memo fields when you are using commands such as APPEND, BROWSE, CHANGE, EDIT, or READ. The WINDOW clause that you specify with the @...SAY...GET command for editing memo fields overrides the window specified with this command.

If you specify SET WINDOW OF MEMO TO without specifying the name of a window, *Visual* dBASE clears any previously defined window name.

Example

The following example uses SET WINDOW OF MEMO to specify a window to use for editing a memo field:

```
DEFINE WINDOW MainWindow FROM 1, 1 TO 15,70
DEFINE WINDOW MemoEdit FROM 6,30 TO 14,68
USE Company EXCLUSIVE
ACTIVATE WINDOW MainWindow
SET WINDOW OF MEMO TO MemoEdit
@ 2, 1 GET Notes OPEN WINDOW MemoEdit
READ
CLOSE DATABASES
DEACTIVATE WINDOW MainWindow
```

Portability

Not supported in dBASE III PLUS.

See Also

ACTIVATE WINDOW, CLEAR WINDOW, DEFINE WINDOW, MOVE WINDOW

SET()

Environment

Returns the current setting of a SET command or function key.

Syntax

SET(<expC>)

<expC> A character expression that is the SET command or function key whose setting value to return.

Description

Use SET() to learn a SET or function key setting so that you can change it or save it. For example, you can issue SET() at the beginning of a program to learn current settings. You can then save these settings in memory variables, change the settings, and restore the original settings from the memory variables at the end of the program.

When dBASE supports a SET and a SET...TO command that use the same keyword, SET() returns the on | off setting and SETTO() returns the SET...TO setting. For example, you can issue SET CARRY ON, SET CARRY OFF, or SET CARRY TO <field list>. SET("CARRY") returns "ON" or "OFF" and SETTO("CARRY") returns the field list as a character expression.

If dBASE supports a SET...TO command but not a corresponding SET command, SET() and SETTO() both return the SET...TO value. For example, SET("BLOCKSIZE") and SETTO("BLOCKSIZE") both return the same value.

When <expC> is a function key name, such as "F4", SET() returns the function key setting. To return the value of a Ctrl+function key setting, add 10 to the function key number; to return the value of a Shift+function key setting, add 20 to the function key number. That is, to return the value of Ctrl+F4, use SET("F14"), and to return the value of Shift+F4, use SET("F24").

If a procedure file is open, SET("PROCEDURE") returns the name of the procedure file. If more than one procedure file is open, SET("PROCEDURE") returns the name of the first one loaded. To return the name of another open procedure file, enter a number as the second argument; for example, SET("PROCEDURE",2) returns the name of the second procedure file that was loaded. If no procedure files are open, SET("PROCEDURE") returns an empty string ("").

The command you specify for <expC> can be abbreviated to four letters in most cases, following the same rules as those for abbreviating keywords. For example, SET("DECT") and SET("DECIMALS") have the same meaning. The <expC> argument is not case-sensitive.

S

Example

In this example, you see a typical use of SET(). The current color settings are obtained using SET() and saved. The colors are changed, some work is performed (in this case, simply clearing the screen) and the original setting is then restored:

```
Oldcolors=SET("ATTRIBUTE")
SET COLOR TO G/B,R/B
```

SET ()

```
CLEAR
SET COLOR TO &Oldcolors
* Old colors restored
```

The following example shows most of the SET() arguments along with the kind of response obtained:

```
? SET("F1")                && help;
? SET("ALTERNATE")          && ON
? SET("ATTRIBUTE")          && RGB+/B,N/W,N/G && N/G,W/B,RGB+/B,B/W,N/W
? SET("AUTO")               && OFF
? SET("BELL")               && ON
? SET("CARRY")              && OFF
? SET("CATALOG")            && OFF
? SET("CENTURY")            && OFF
? SET("CONFIRM")            && OFF
? SET("CONSOLE")            && ON
? SET("COVER")              && OFF
? SET("CUAENTER")           && ON
? SET("CURRENCY")           && LEFT
? SET("DELIMITER")          && OFF
? SET("DELETED")            && OFF
? SET("DESIGN")             && ON
? SET("ECHO")               && OFF
? SET("ENCRYPTION")         && OFF
? SET("ESCAPE")             && ON
? SET("EXACT")              && OFF
? SET("EXCLUSIVE")          && OFF
? SET("FIELDS")             && OFF
? SET("FORMAT")             && ""
? SET("FULLPATH")           && OFF
? SET("HEADING")            && ON
? SET("HELP")               && ON
? SET("INTENSITY")          && ON
? SET("LIBRARY")            && ""
? SET("LOCK")               && ON
? SET("MARGIN")             && 0
? SET("MARK")               && /
? SET("MEMOWIDTH")          && 50
? SET("MESSAGE")            && ""
? SET("NEAR")               && OFF
? SET("ODOMETER")           && 100
? SET("ORDER")              && ""
? SET("PATH")               && ""
? SET("PCOL")               && 0
? SET("POINT")              && .
? SET("PRECISION")          && 16
? SET("PRINTER")            && OFF
? SET("RELATION")           && ""
? SET("REPROCESS")          && 0
? SET("SAFETY")             && OFF
? SET("SEPARATOR")          && ,
? SET("SKIP")               && ""
? SET("SPACE")              && ON
? SET("STEP")               && OFF
? SET("TALK")               && ON
```



```
? SET("TOPIC")           && ""
? SET("TYPE")             && 50
? SET("UNIQUE")           && OFF
```

Portability

Not supported in dBASE III PLUS.

See Also

DISPLAY STATUS, SET, SETTO()

SETTO()

Environment

Returns the current setting of a SET...TO command or function key.

Syntax

SETTO(<expC>)

<expC> A character expression that is the SET...TO command whose setting value to return.

Description

Use SETTO() to learn a SET or function key setting so that you can change it or save it. For example, you can issue SETTO() at the beginning of a program to learn current settings. You can then save these settings in memory variables, change the settings, and restore the original settings from the memory variables at the end of the program.

When dBASE supports a SET and a SET...TO command that use the same keyword, SET() returns the SET setting and SETTO() returns the SET...TO setting. For example, you can issue SET CARRY ON, SET CARRY OFF, or SET CARRY TO <field list>. SET("CARRY") returns the ON or OFF setting and SETTO("CARRY") returns the field list as a character expression.

SETTO() is almost identical to SET(). For more information, see SET().

Example

In this example, you see a typical use of SETTO() when printing a special report to a particular printer. The current setting of SET PRINTER TO is stored. SET PRINTER TO is set to LPT2 and a report is printed. The original setting is then restored:

```
Oldprintto=SETTO("PRINTER")  && e.g. LPT1
SET PRINTER TO LPT2
REPORT FORM Myreport TO PRINTER
SET PRINTER TO &Oldprintto
```

The following example shows most of the SETTO() arguments along with the kind of response obtained:

SETTO()

? SETTO("ALTERNATE")	&& D:\VISUALDB\EXAMPLES\SETTO().RES
? SETTO("BORDER")	&& SINGLE
? SETTO("BELL")	&& 512,2
? SETTO("BLOCK")	&& 1
? SETTO("CATALOG")	&&
? SETTO("CURRENCY")	&& \$
? SETTO("DATE")	&& MDY
? SETTO("DECIMALS")	&& 2
? SETTO("DELIMITER")	&& ::
? SETTO("DEVICE")	&& SCREEN
? SETTO("DIRECTORY")	&& D:\VISUALDB\EXAMPLES
? SETTO("DISPLAY")	&& EGA25
? SETTO("EDITOR")	&&
? SETTO("FILTER")	&&
? SETTO("FIELDS")	&&
? SETTO("FORMAT")	&&
? SETTO("HELP")	&& DBASEWIN.HLP
? SETTO("IBLOCK")	&& 1
? SETTO("LIBRARY")	&&
? SETTO("MARGIN")	&& 0
? SETTO("MARK")	&& /
? SETTO("MEMOWIDTH")	&& 50
? SETTO("MESSAGE")	&&
? SETTO("ORDER")	&&
? SETTO("PATH")	&&
? SETTO("PCOL")	&& 0
? SETTO("POINT")	&& .
? SETTO("PRECISION")	&& 16
? SETTO("PROCEDURE")	&&
? SETTO("PROW")	&& 0
? SETTO("PRINTER")	&& LPT1
? SETTO("RELATION")	&&
? SETTO("REFRESH")	&& 0
? SETTO("RETRACE")	&& OFF
? SETTO("SEPARATOR")	&& ,
? SETTO("SKIP")	&&
? SETTO("TOPIC")	&&

Portability

Not supported in dBASE IV or dBASE III PLUS.

See Also

DISPLAY STATUS, SET, SET()

SHELL()

Environment

Removes or restores the dBASE interactive environment (known as the shell) when a form is open. Returns a logical value corresponding to the previous SHELL() state.

Syntax

SHELL([<expl1>], [<expl2>])

<expl1> The value that determines whether to hide or restore the shell. If <expl1> is false (.F.), dBASE hides the shell.

<expl2> The value that determines whether the dBASE Application window remains visible when the shell is hidden. If <expl2> is true (.T.), the Application window is visible. If <expl1> is true, the full shell is on and <expl2> is ignored. If you open an MDI form, the Application window stays visible (to contain the form) regardless of the <expl2> value.

Description

Use SHELL(.F.) in programs to temporarily hide the standard dBASE environment, allowing your application to take control of the user's working environment. To restore the dBASE interactive environment, issue SHELL(.T.). The environment is also restored when the user closes the form that SHELL() is activated for.

SHELL(.F., .F.) operates differently when you are working in a form that is defined as a top-level MDI form (formname.MDI=.F.) or in a form that is not a top-level MDI form (formname.MDI=.T.).

- When formname.MDI=.F. for the active form, SHELL(.F., .F.) appears to remove dBASE from the user's system. The form name becomes the application name that appears in the Windows Task List in place of "Visual dBASE." This makes your application look like a standalone application, and is the typical use for SHELL().
- When formname.MDI=.T. for the active form, the menu system associated with the form appears as the menu at the top of the screen instead of at the top of the form. The user remains in dBASE, but the dBASE menu is replaced by the menu defined by the active form. However, the user can still access the SpeedBar if it is active. The user can click in the Command window to close the form.

Using SHELL() in a program has the same effect as changing the Visible property of the FrameWin object of _app, as shown in the following example. For more information, see _app.

```
SHELL(.F.)  && same effect as next line
_app.FrameWin.Visible = .F.
```

If you issue SHELL(.F.) in the Command window, you exit to Windows momentarily and then return to dBASE.

Example

This example shows the code generated by the form designer for a Shell Test form that simply sets the left double click button to SHELL(.t.) and the right double click button to SHELL(.f.).

SHOW MENU

```
LOCAL f
f = NEW SHELL ()
f.Open()

CLASS SHELL OF FORM
    this.OnRightDbClick = {shell(.t.)}
    this.OnLeftDbClick = {shell(.f.)}
    this.EscExit = .T.
    this.mdi = .f.
    * set mdi=.t. to see effect with mdi
    this.Text = "Shell Test .t."
    this.Width = 48.00
    this.Top = 2.00
    this.Left = 2.00
    this.Height = 15.00
    this.Minimize = .F.
    this.Maximize = .F.
ENDCLASS
```

When the form is activated e.g. with DO Shell.wfm, the Shell Test form appears on the screen (with mdi=.f.). Double clicking with the left mouse button makes other windows in the dBASE screen disappear. Double clicking with the right button makes them reappear. * When you set mdi=.t., the Shell Test form can be accessed by pressing Ctrl Tab to show the Windows windows.

Portability

Not supported in dBASE IV or dBASE III PLUS.

See Also

_app, DEFINE, QUIT, SET DESIGN

SHOW MENU

dBASE IV menus

Displays, but does not enable, a previously-defined dBASE IV menu bar. This command is supported primarily for compatibility with dBASE IV. In *Visual* dBASE, use DEFINE, OPEN FORM, and READMODAL() to create and activate menus associated with forms.

For complete syntax information on SHOW MENU, see online Help. For information about defining forms, see the Forms chapters in the *User's Guide*.

SHOW OBJECT

Objects

Updates the appearance of an object to reflect its most recent property settings.

Syntax

SHOW OBJECT *<container reference>.<object reference>*

<container reference>.<object reference> *<container reference>* is an object reference variable pointing to the object (usually a form) that contains the object. *<object reference>* is an object reference variable pointing to the object itself.

Description

Use SHOW OBJECT to refresh an object on the screen. For example, when you change a bitmap image on a push button with the UpBitMap property, SHOW OBJECT makes the new image appear in the push button.

Example

The following example defines a form with two pushbuttons that move the record pointer. An OnClick property calls a procedure that evaluates record pointer position with regard to EOF() or BOF(). If the record pointer is located at EOF() or BOF(), the procedure changes properties of the buttons and uses SHOW OBJECT to refresh the displayed button as per the new properties. The ELSE clause of the procedure refreshes the pushbutton to original property values and moves the record pointer:

```
USE ANIMALS
SET PROCEDURE TO PROGRAM(1) ADDITIVE
DEFINE FORM F1;
    PROPERTY Text "SHOW OBJECT Command Demo"
DEFINE PUSHBUTTON P1 OF F1 AT 9,15;
    PROPERTY Text "Previous",;
    Width 12, Height 2,;
    OnClick Prev
DEFINE PUSHBUTTON P2 OF F1 AT 12,15;
    PROPERTY Text "Next",;
    Width 12, Height 2,;
    OnClick Next
OPEN FORM F1

PROCEDURE Prev
    IF BOF()
        Form.P1.Text = "You are at BOF"
        Form.P1.Width = 16
        Form.P1.Left = 12
        SHOW OBJECT form.P1
    ELSE
        Form.P1.Text = "Previous"
        Form.P1.Width = 10
        Form.P1.Left = 15
        SHOW OBJECT form.P1
    SKIP-1
ENDIF
RETURN
```

SHOW POPUP

```
PROCEDURE Next
  IF EOF()
    Form.P2.Text = "You are at EOF"
    Form.P2.Width = 16
    Form.P2.Left = 12
    SHOW OBJECT form.P2
  ELSE
    Form.P2.Text = "Next"
    Form.P2.Width = 10
    Form.P2.Left = 15
    SHOW OBJECT form.P2
  SKIP
ENDIF
RETURN
```

Portability

Not supported in dBASE IV or dBASE III PLUS.

See Also

UpBitMap

SHOW POPUP

dBASE IV menus

Displays, but does not enable, a previously-defined dBASE IV pop-up menu. This command is supported primarily for compatibility with dBASE IV. In *Visual dBASE*, use `DEFINE`, `OPEN FORM`, and `READMODAL()` to create and activate menus associated with forms.

For complete syntax information on `SHOW POPUP`, see online Help. For information about defining forms, see the Forms chapters in the *User's Guide*.

SIGN()

Numeric data

Returns an integer that indicates if a specified number is positive, negative, or zero (0).

Syntax

`SIGN(<expN>)`

<expN> The numeric or float number whose sign (positive, negative, or zero) to determine.

Description

Use `SIGN()` to determine if a numeric or float expression evaluates to a positive, negative, or zero (0) value. `SIGN()` returns 1 if a specified number is positive, -1 if that number is negative, and 0 if that number is 0.

`SIGN()` always returns an integer, regardless of the value of `SET DECIMALS`.

Example

The following examples illustrate the three possible returned values of SIGN():

```
x = -1234.45
y = 1234.45
z = 0
? SIGN(x)           && Returns -1
? SIGN(y)           && Returns 1
? SIGN(z)           && Returns 0
```

The following example uses SIGN() to display those records in the Clients table that have zero or positive balances in the StartBal field:

```
USE Clients
? "Companies with a starting balance of $00.00"
?
SCAN FOR SIGN(StartBal)=0
? Company, StartBal
ENDSCAN
?
? "Companies with a starting balance greater " + "than $00.00"
?
SCAN FOR SIGN(StartBal)=1
? Company, StartBal
ENDSCAN
CLOSE DATABASES
```

Portability

Not supported in dBASE III PLUS.

See Also

ABS(), MAX(), MIN(), SET DECIMALS

SIN()

Numeric data

Returns the trigonometric sine of an angle.

Syntax

SIN(<expN>)

<expN> The size of the angle in radians. To convert an angle's degree value to radians, use DTOR(). For example, to find the sine of a 30-degree angle, use SIN(DTOR(30)).

S

Description

SIN() calculates the ratio between the side opposite an angle and the hypotenuse in a right triangle. SIN() returns a float from -1 to +1. SIN() returns zero when <expN> is zero, pi, or 2pi radians.

Use SET DECIMALS to set the number of decimal places SIN() displays.

The cosecant of an angle is the reciprocal of the sine of the angle. To return the cosecant of an angle, use 1/SIN().

Example

Following are some ways to use SIN():

? SIN(PI())	&& Returns 0
? SIN(PI()/2)	&& Returns 1
? SIN(DTOR(30))	&& Returns 0.5
? SIN(-3*PI()/2)	&& Returns 1
? -SIN(3*PI()/2)	&& Returns 1

The following program graphs a sine wave. It first draws the x- and y-axes, then plots the points SIN() returns:

```
* Sine.prg
SET TALK OFF
CLEAR
* Draw the x and y axes
@ 11,0 SAY REPLICATE("-", 78)
FOR i = 0 TO 23
    @ i,40 SAY "|"
NEXT
@ 11,40 SAY "."

FOR i = 0 TO 79
    y = 11 - 11 * SIN(i * PI()/20)
    @ y,i SAY "."
NEXT
@ 23,0      && Move cursor below graph
SET TALK ON
```

Portability

Not supported in dBASE III PLUS.

See Also

ASIN(), COS(), DTOR(), PI(), RTOD(), SET DECIMALS, TAN()

SKIP

Fields and records

Moves the record pointer in the current or specified work area.

Syntax

```
SKIP
[<expN>]
[IN <alias>]
```

<expN> The number of records *Visual* dBASE moves the record pointer forward or backward in the table open in the current or specified work area. If <expN> evaluates to a negative number, the record pointer moves backward. SKIP with no <expN> argument moves the record pointer forward one record.

IN <alias> Specifies the work area in which to move the record pointer. You can specify a work area number (1 through 225), letter (A through J), or alias name. The work area letter or alias name must be enclosed in quotes. SKIP without IN <alias> moves the record pointer in the current work area.

Description

Use SKIP to move the record pointer relative to its current position. For an unindexed table, the record pointer moves a specified number of records. For an indexed table, the record pointer skips the specified number of records following the index record order.

If you issue a SKIP command when the record pointer is at the last record in a table, EOF() returns .T. Issuing any additional SKIP commands returns an error. Similarly, if you issue a SKIP -1 command when the record pointer is at the first record of a file, BOF() returns .T., and a subsequent negative SKIP command returns an error.

SKIP IN <alias> lets you advance the record pointer in another work area without selecting that work area first with the SELECT command.

Example

The following example uses SKIP to move the record pointer through the records of a table to display or print a listing of selected fields:

```
SET SAFETY OFF
SET TALK OFF
USE Country EXCLUSIVE
INDEX ON GNP TAG GNP
? CENTER("Country List-Lowest GNP First")
?
DO WHILE .NOT. EOF()
    ? Name AT 2, GNP AT 20, Capital AT 40
    SKIP
ENDDO
CLOSE ALL
```

SKIP accompanied with a negative expression can be used to accomplish the reverse of the previous example:

```
USE Country EXCLUSIVE
INDEX ON Population TAG Pop
? CENTER("Country List-Most Populous First")
?
GO BOTTOM
DO WHILE .NOT. BOF()
    ? Name AT 2, Population AT 20, Capital AT 40
    SKIP-1
ENDDO
CLOSE ALL
SET TALK ON
SET SAFETY ON
RETURN
```

See Also

ALIAS(), BOF(), EOF(), GO, SCAN

Pauses a program for a specified interval or until a specified time.

Syntax

```
SLEEP <seconds expN> |  
UNTIL <time expC> [, <date expC>]
```

<seconds expN> The number of seconds to pause the program. The numeric expression must evaluate to an integer ranging from 1 to 65,535 (1 second to about 18 hours). Counting starts from the time you issue the SLEEP command.

UNTIL <time expC> Causes program execution to pause until a specified time (<time expC>) on the current day. If you also specify <date expC>, the program pauses until the time on that day. The time and date dBASE uses are the system time and date. You can set the system time with SET TIME and the system date with SET DATE TO. If the time has already passed, SLEEP UNTIL <time expC> has no effect.

The <time expC> argument is a character expression that must evaluate to a time in HH<delimiter>MM<delimiter>SS (24-hour) format. A typical format for <time expC> is "HH:MM:SS". The delimiter is conventionally a colon but can be any other single keyboard character except a number. HH is a 1- or 2-digit number of hours, MM a 1- or 2-digit number of minutes, and SS a 1- or 2-digit number of seconds.

<date expC> An optional date until which the program is to pause. The <date expC> argument is a character expression (*not* a date expression) that must evaluate to a date in MM<delimiter>DD<delimiter>YY format if SET DATE is AMERICAN. (The typical format is "MM/DD/YY".) The delimiter for <date expC> is conventionally a forward slash (/) but can be any other single keyboard character except a number. MM is a one- or two-digit number of months, DD a one- or two-digit number of days, and YY a one- or two-digit number of years. If the date has already passed, SLEEP UNTIL <time expC> [, <date expC>] has no effect. If you want to specify a value for <date expC>, you must also specify a value for <time expC>.

Description

Use SLEEP to pause a program either for <seconds expN> seconds or until a specified time (<time expC>). The specified time is the same day the program is running unless you specify a date with <date expC>. If SET ESCAPE is ON, you can interrupt SLEEP by pressing *Esc*.

Note If SET ESCAPE is OFF, there is no way to interrupt SLEEP. However, you can use *Ctrl+Esc* and *Alt+Tab* to switch to another Windows application, or *Alt+F4* to exit dBASE.

Although SLEEP can generate a pause from the Command window, programmers use it primarily within programs. For example, you can use SLEEP to generate a pause between multiple displaying windows or to allow a user to read a message on the screen or complete an action. Pauses are also useful when you need to delay program execution until a specific time.

SLEEP is an alternative to using a DO WHILE loop, a FOR...NEXT loop, or WAIT to generate pauses in a program. SLEEP is more accurate than using loops because it's

independent of the execution speed of the system. You can also use INKEY(<expN>) if you want the user to be able to interrupt the pause and continue with program processing.

Example

The following example uses SLEEP to delay execution for five seconds:

```
SET ESCAPE ON
* ESCAPE ALLOWS YOU TO ABORT THE SLEEP COMMAND
SLEEP 5
```

The next example uses SLEEP to delay execution until 7:30 p.m. on the same day the program is running:

```
SET ESCAPE ON
* ESCAPE ALLOWS YOU TO ABORT THE SLEEP COMMAND
SLEEP UNTIL "19:30:00"
```

The last example uses SLEEP to delay execution until 5:30 p.m.: on May 31, 1997 (SET DATE is AMERICAN):

```
SET ESCAPE ON
* ESCAPE ALLOWS YOU TO ABORT THE SLEEP COMMAND
SLEEP UNTIL "17:30:00", "05/31/97"
```

Portability

Not supported in dBASE IV or dBASE III PLUS.

See Also

DO WHILE, FOR...NEXT, INKEY(), SET DATE TO, SET TIME, WAIT

SORT

Table organization

Copies the current table to a new table, arranging records in the specified order.

Syntax

```
SORT TO <filename> | ?
[[TYPE] PARADOX | DBASE]
ON <field 1> [/A | /D [/C]]
    [, <field 2> [/A | /D [/C]]...]
[<scope>]
[FOR <condition 1>]
[WHILE <condition 2>]
[ASCENDING | DESCENDING]
```

<filename> | ? The new table file to copy and sort the current table's records to. By default, *Visual* dBASE assigns a .DBF extension to <filename> and saves the file in the current directory. The ? option displays a dialog box, in which you specify the name of the target file and the directory to save it in.

You can also create a table in a database (defined using the BDE Configuration Utility) by specifying the database as a prefix (enclosed in colons) to the name of the table, that is, *:database name:table name*. If the database is not already open, *Visual* dBASE displays a dialog box, in which you specify the parameters, such as a login name and password, necessary to establish a connection to that database.

[TYPE] PARADOX | DBASE Specifies the type of table you want to create. The TYPE keyword is included for readability only; it has no effect on the operation of the command.

Specifying DBASE creates a dBASE table (the default). If you don't include an extension for *<filename>*, *Visual* dBASE assigns a .DBF extension.

Specifying PARADOX creates a Paradox table with a .DB extension.

ON <field 1> Makes *<field 1>* the first field of *<filename>* and sorts *<filename>* records by the values in *<field 1>*, which can be any data type except binary, memo, or OLE.

/A Sorts records in ascending order (A to Z; 1 to 9; past to future; false then true). Since this is the default sort order, include /A for readability only.

/D Sorts records in descending order.

/C Removes the distinction between uppercase and lowercase letters. When you specify both A and C, or both D and C, use only one forward slash (for example, /AC).

<field 2> [/A | /D [/C]] ... Sorts on a second field so that the new table is ordered first according to *<field 1>*, then, for identical values of *<field 1>*, according to *<field 2>*. If a third field is specified, records with identical values in *<field 1>* and in *<field 2>* are then sorted according to *<field 3>*. The sorting continues in this way for as many fields as are specified.

<scope> The number of records to copy from the current table to *<filename>* and sort. RECORD *<n>* identifies a single record by its record number. NEXT *<n>* identifies *n* records, beginning with the current record. ALL specifies all records. REST specifies all records from the current record to the end of the file.

FOR <condition 1>

WHILE <condition 2> Determines which records are affected by SORT. FOR restricts SORT to records that meet *<condition 1>*. WHILE starts with the current record and continues with each subsequent record as long as *<condition 2>* is true.

ASCENDING Sorts all specified fields for which you don't include a sort order in ascending order. Since this is the default, include ASCENDING for readability only.

DESCENDING Sorts all specified fields for which you don't include a sort order in descending order.

Description

The SORT command creates a new table in which the records in the current table are positioned in the order of the specified key fields. Records marked for deletion are ignored if SET DELETED is ON.

When you use SORT, dBASE creates a temporary index file. During the sorting process, your disk must have space for this temporary index file and the new table file.

SORT differs from INDEX in that it creates a new table rather than provide an index to the original table. Although using SORT is generally not as efficient as using an index to organize tables, you might want to use SORT for the following applications:

- To archive an outdated table and store it in a sorted order
- To create a table that is a sorted subset of an original table
- To maintain a small table that needs to be sorted in only one order
- To create an ordered table where record numbers are sequential and contiguous

You might also want to sort a table according to the order of an index file that it commonly uses if the data in the table doesn't change frequently, since *Visual dBASE* can access a table sorted in index order much faster than an unsorted table. In a table sorted in index order, sequential records are next to each other on the disk.

Example

The following example uses SORT to make new tables:

```
USE Clients
SET DELETED ON
* remove the deleted records
SORT TO ClientC ON Company /C
USE ClientC
BROWSE FIELDS Company;
    TITLE "Sorted by Company, case ignored"
```

ClientC is sorted by Company ignoring upper- and lowercase.

```
USE Clients
SORT TO ClientP ;
    ON State_Prov , City /D, Company /DC;
    FOR Zip_postal = "9" TYPE PARADOX
* Sort:
*   State Ascending by default
*   City Descending
*   Company descending and ignoring case
*   Only Zip_postal codes beginning with 9
*   Creating a paradox table
SET DBTYPE TO PARADOX
USE ClientP
BROWSE;
    FIELDS Company, City, State_Prov, Zip_postal ;
    TITLE "Paradox file"
SET DBTYPE TO                                && reset to dBASE
```

See Also

INDEX

SOUNDEX()

Returns a four-character string that represents the SOUNDEX (sound-alike) code of another string.

Syntax

SOUNDEX(<expC> | <memo field>)

<expC> | <memo field> The string or memo field for which to calculate the soundex code. The string or memo can be any word or nonword that is a particular sound when spoken. The string or memo can also be more than one word or nonword with spaces between the words.

Description

SOUNDEX() returns a four-character code that represents the phonetic value of a character expression or memo field. The code is in the form "letter digit digit digit," where "letter" is the first alphabetic character in the expression being evaluated. The more phonetically similar two strings are, the more similar their SOUNDEX codes.

Use SOUNDEX() to find words that sound similar, or are spelled similarly, such as names like "Smith," "Smyth," and "Smythe." Using the U.S. language driver, these all evaluate to S531. You can index a table on the SOUNDEX() value of a field, then use FIND or SEEK with SOUNDEX() for names that users want to search for. For example, if a user wants to search for "Smith," convert "Smith" to SOUNDEX("Smith"), then search for that code in a table with a master index based on SOUNDEX(lastname). You can also use SOUNDEX() with LOCATE.

SOUNDEX() returns "0000" if the character expression or memo field is an empty string or if the first nonblank character isn't a letter. SOUNDEX() returns 0's for the first digit encountered and for all following characters, regardless of whether they're digits or alphabetic characters.

To compare the SOUNDEX values of two character expressions or memo fields, use DIFFERENCE(). If you want to compare the character-by-character similarity between two strings rather than the phonetic similarity, use LIKE().

SOUNDEX() is language driver-specific. For more information on language drivers, see Appendix C in the *Programmer's Guide*.

If the current language driver is U.S., SOUNDEX() does the following to calculate the phonetic value of a string:

- Ignores leading spaces.
- Ignores the letters A, E, I, O, U, Y, H, and W.
- Ignores case.
- Converts the first nonblank character to uppercase and makes it the first character in the SOUNDEX code.
- Converts B, F, P, and V to 1.

- Converts C, G, J, K, Q, S, X, and Z to 2.
- Converts D and T to 3.
- Converts L to 4.
- Converts M and N to 5.
- Converts R to 6.
- Removes the second occurrence of any adjacent letters that receive the same digits as phonetic values.
- Pads the end of the resulting string with zeros if fewer than three digits remain.
- Truncates the resulting string to three digits if more than three digits remain.
- Concatenates the first character of the code to the remaining three digits to create the "letter digit digit digit" soundex code.

Example

The following example uses SOUNDEX() to get a code value for the sound of each text string:

? SOUNDEX("bo")	&& Returns B000
? SOUNDEX("beau")	&& Returns B000
? SOUNDEX("bow")	&& Returns B000
? SOUNDEX("bow") = SOUNDEX("beau")	&& Returns .T.
? SOUNDEX("Conrad")	&& Returns C563
? SOUNDEX("")	&& Returns 0000
? SOUNDEX("1dBASEWay")	&& Returns 0000
? SOUNDEX("Go5ldeñArches")	&& Returns G000
? SOUNDEX("Go15deñArches")	&& Returns G400

The next example uses SOUNDEX() when displaying the contents of a field:

```
USE Clients
? TRIM(City) + " returns a value of " + SOUNDEX(City);    && Returns "Atlanta returns ;
                                                         a value of A345"
CLOSE DATABASES
```

Portability

Not supported in dBASE III PLUS. The <memo field> argument isn't supported in dBASE IV.

See Also

DIFFERENCE(), FIND, INDEX, LIKE(), LOCATE, SEEK, SET EXACT

SPACE()

Returns a specified number of space characters.

Syntax

SPACE(<expN>)

<expN> The number of spaces to return.

Description

SPACE() returns a character string composed of a specified number of space characters. The space character is ASCII code 32. The largest number of spaces you can specify is 32766, the maximum length of a string.

If <expN> is 0, SPACE() returns an empty string. If <expN> is less than 0, dBASE displays an error.

To create a string using a character other than space, use REPLICATE().

Example

The following example uses SPACE() to initialize character memory variables of a specified length.

mCompany=SPACE(20)	&& mCompany contains 20 spaces
STORE SPACE(20) TO mCompany	&& same as above

The following example uses SPACE() to create spacing in output text. The example would return a trimmed city name, a comma and space, a two-letter State code, five spaces and the Zipcode.

```

USE Clients
SCAN
? Company
? Contact
? Address
? TRIM(City) + ", " + State_Prov + SPACE(5);
+ Zip_P_Code
?
ENDSCAN

```

Portability

Both dBASE IV and dBASE III PLUS limit the return value of SPACE() to 254 characters.

See Also

ASC(), CHR(), REPLICATE()

SQLERROR()

Error handling and debugging

Returns the number of the last server error.

Syntax

SQLERROR()

Description

Use SQLERROR() to determine the error number of the last server error. To learn the text of the error message itself, use SQLMESSAGE().

See the table in the description of ERROR() that compares ERROR(), MESSAGE(), DBERROR(), DBMESSAGE(), SQLERROR(), SQLMESSAGE(), and CERROR().

See online Help for a listing of all error messages.

Example

The following example uses SQLERROR() and SQLMESSAGE() to return an SQL error number and SQL error message to an ON ERROR routine that displays a MDI form with an error report:

```
ON ERROR DO ErrHndlr WITH ERROR(), MESSAGE(), ;
    SQLERROR(), SQLMESSAGE(), PROGRAM(), LINENO()
SET DBTYPE TO DBASE
OPEN DATABASE CAClients
SET DATABASE TO CAClients
errorCode = SQLEXP("SELECT Company, City ;
    FROM Company WHERE State_Prov='CA'", "StateCA.DBF")
IF errorCode = 0
    SET DATABASE TO
    USE StateCa
    LIST
ENDIF
RETURN

PROCEDURE ErrHndlr
PARAMETERS nErrorNo, cErrMess, nSQLErrorNo, cSQLErrMess, cProgram, nLineNo
DEFINE FORM HeadsUp FROM 10,20 TO 20,55;
    PROPERTY Text "Heads Up"
DEFINE TEXT Line1 OF HeadsUp AT 2,10 ;
    PROPERTY Text "An Error has occurred", Width 24, ColorNormal "R+/W"
DEFINE TEXT Line2 OF HeadsUp AT 4,2;
    PROPERTY Text ;
    IIF(ERROR()=240,cSqlErrMess,cErrMess), Width 33
DEFINE TEXT Line3 OF HeadsUp AT 5,2;
    PROPERTY Text "Number: " + ;
    IIF(ERROR()=240,STR(nSQLErrorNo),STR(nErrorNo)), Width 24
DEFINE TEXT Line4 OF HeadsUp AT 6,2;
    PROPERTY Text "Program: " + cProgram, Width 22
DEFINE TEXT Line5 OF HeadsUp AT 7,2;
    PROPERTY Text "Line #: " + STR(nLineNo), Width 22
OPEN FORM HeadsUp
```

Portability

Not supported in dBASE IV or dBASE III PLUS.

See Also

CERROR(), DBERROR(), DBMESSAGE(), ERROR(), MESSAGE(), ON ERROR, RETRY, SQLMESSAGE()

SQLEXEC()**Table basics**

Executes an SQL statement in the current database or on specified dBASE or Paradox tables.

Syntax

SQLEXEC(<SQL statement expC> [,<Answer table expC>])

<SQL statement expC> A character string that contains an SQL statement. The SQL statement must follow server-specific dialect rules for the current database and must be enclosed in quotes. For Paradox and dBASE tables, the dialect is the same as that used by the Borland InterBase database server, which is ANSI-compliant. Character strings and SQL or IDAPI reserved words contained within the SQL statement must also be enclosed in either single or double quotes. (Single quotes are normally used.).

<Answer table expC> Paradox or dBASE table that stores the data returned by an SQL SELECT statement; must also be in quotes. If you specify a file without including its path, *Visual* dBASE creates the file in the current directory, then in the path you specify with SET PATH. If you specify a file without including its extension, *Visual* dBASE assumes the default table type specified with the SET DBTYPE command. If you don't specify a table name, *Visual* dBASE creates a table named Answer with the extension defined by the current DBTYPE setting.

You can also specify the name of an already open database (defined for a file directory location only) as a prefix (enclosed in colons) to the name of the answer table, that is, *:database name:table name*. You cannot specify the location of an answer table on a database server.

Description

SQLEXEC() executes a SQL statement in the current database set by SET DATABASE, or if a database is not set, on tables in the current or a specified directory. (You can preface the name of a table with its directory location or specify an already open database by enclosing the database name in colons, for example, *:database name:table name*. If you're using Borland SQL Link to connect to a database server, *Visual* dBASE passes the SQL statement you specify directly to the database server where the database selected by SET DATABASE resides.

When an SQL statement contains SQL or IDAPI reserved words and you are executing the statement on dBASE or Paradox tables, you need to enclose the reserved words in single(') or double(") quotes and use SQL table aliases (different than the aliases associated with dBASE tables) to identify fields, for example:

```
SELECT * FROM company.dbf b WHERE b.'CHAR' = 'element'
```

You can use table aliases to qualify fields specified in the `SELECT`, `WHERE`, `GROUP BY`, or `ORDER BY` clauses of `SELECT` statements. This is particularly useful when querying data from more than one table.

`SQLEXEC()` returns error codes with the same values as those returned by `ERROR()` and `MESSAGE()`; a value of zero indicates that no error occurred as a result of the statement's execution. If an error occurs, you can use `DBERROR()` and `DBMESSAGE()` functions to return IDAPI errors or use the `SQLERROR()` and `SQLMESSAGE()` functions to obtain information directly from the database server about the cause of an error. (Also, the `ERROR()` function returns an error code of 240 if a server error occurs.)

Example

The following example executes an SQL `SELECT` statement on the server table `Company`:

```
SET DBTYPE TO DBASE
OPEN DATABASE CAClients
SET DATABASE TO CAClients
errorCode = SQLEXEC("SELECT Company, City ;
    FROM Company WHERE State_Prov='CA'", "StateCA.DBF")
IF errorCode = 0
    SET DATABASE TO
    USE StateCa
    LIST
ENDIF
RETURN
```

Portability

Not supported in dBASE IV or dBASE III PLUS.

See Also

`DBERROR()`, `DBMESSAGE()`, `ERROR()`, `MESSAGE()`, `OPEN DATABASE`, `SET DATABASE`, `SET DBTYPE`, `SET PATH`, `SQLERROR()`, `SQLMESSAGE()`

SQLMESSAGE()

Error handling and debugging

Returns the most recent server error message.

Syntax

`SQLMESSAGE()`

Description

Use `SQLMESSAGE()` to determine the error message of the last server error. To learn the error code, use `SQLERROR()`.

See online Help for a listing of all error messages.

Example

See `SQLERROR()` for an example of using `SQLMESSAGE()`.

Portability

Not supported in dBASE IV or dBASE III PLUS.

See Also

`CERROR()`, `DBERROR()`, `DBMESSAGE()`, `ERROR()`, `MESSAGE()`, `ON ERROR`, `RETRY`, `SQLERROR()`

SQRT()

Numeric data

Returns the square root of a number.

Syntax

`SQRT(<expN>)`

<expN> A positive number whose square root to return. If **<expN>** is a negative number, dBASE returns an error.

Description

`SQRT()` returns as a float the positive square root of a non-negative number. For example `SQRT(36)` returns 6 because $6^2=36$. The square root of 0 is 0.

An alternate way to find the square root is to raise the value to the power of 0.5. For example, the following two commands return the same value:

```
? SQRT(36)                && returns 6.00
? 36^.5                   && returns 6.00
```

Use `SET DECIMALS` to set the number of decimal places `SQRT()` displays.

Example

The following example uses `SQRT()` to compute the length of a rafter after room width and rise to the peak have been entered by the user:

```
SET PROCEDURE TO PROGRAM(1) ADDITIVE
LOCAL f
f=NEW Rafter()
f.OPEN()
CLASS Rafter OF FORM
  this.Top=2
  this.Left=2
  this.Width=30
  this.Height=12
  this.Text = "Rafter Computer"
DEFINE ENTRYFIELD Room OF THIS AT 2,22;
  PROPERTY Picture "999",Value 0, Width 4
DEFINE ENTRYFIELD Rise OF THIS at 4,22;
  PROPERTY Picture "99",Value 0, Width 4
DEFINE ENTRYFIELD RaftLength OF THIS AT 6,22;
```

```

    PROPERTY Width 6, Value 0, OnGotFocus Results
DEFINE TEXT Ln1 OF THIS AT 2,3;
    PROPERTY Text "Enter width of room:", Width 15
DEFINE TEXT Ln2 OF THIS AT 4,3;
    PROPERTY Text "Enter height of rise:", Width 15
DEFINE TEXT Ln3 OF THIS AT 6,3;
    PROPERTY Text "Cut rafter to this length:", Width 18, ColorNormal "R/W"
DEFINE PUSHBUTTON Exit OF THIS AT 9,11;
    PROPERTY TEXT "Exit", OnClick {;Form.Close()}
ENDCLASS

FUNCTION Results
Form.RaftLength.Value= STR(SQRT(((Form.Room.Value/2)^2)+(Form.Rise.Value^2)),5,2)
RETURN .T.

```

See Also

EXP(), LOG(), LOG(10), SET DECIMALS

STATIC

Memory variables

Declares local memory variables that you can use only in the subroutine where they're declared but whose values remain in memory until you exit dBASE.

Syntax

STATIC <variable 1> [= <value 1>] [,<variable 2> [= <value>] ...]

<variable> The variable to declare static.

<value> The value to assign to the variable.

Description

Use STATIC to declare memory variables available only to a particular subroutine but public in memory duration. Static variables are different from other types of memory variables in two important ways:

- You can declare and assign a value to a static variable in a single statement.
- Static variables initialized in a single statement are assigned the initialization value only the first time the subroutine is run.

When you declare a variable STATIC in a subroutine—a program, procedure, or user-defined function (UDF)—that variable is visible only within that subroutine. It is not visible to higher- or lower-level subroutines. However, when the subroutine ends, the static variable remains in memory, with the value it had when the subroutine ended. If the subroutine is called again, the variable has the value it had when the subroutine last ended, and keeps that value until the subroutine changes it.

When you declare a variable STATIC without assigning it a value, dBASE creates the variable and assigns it the value .F.

Because static variables are not released when the subroutine in which they are created ends, you can use them to retain values for subsequent times that subroutine runs. To do this, declare and initialize the variable in a single statement, as shown below:

S

```

** subroutine mtest
  STATIC mvar = 100

```

The first time dBASE encounters this statement, mvar is initialized to a value of 100. If the subroutine mtest is run again, mvar is not reinitialized to a value of 100. Instead, mvar retains whatever value it had when mtest last ended.

See PUBLIC for a table that compares the scope and availability of public, private, local, and static variables. See Chapter 5 in the *Programmer's Guide* for more information on initializing and retaining values of static variables.

Example

The following example branches from a main program to several lower-level procedures to demonstrate that STATIC variables are available only at the program level in which they are declared and that they hold that value even if the variable is changed in another procedural level:

```

* **Main.Prg**
CLOSE ALL
CLEAR ALL
CLEAR
SET TALK OFF
? ***Main.PRG***
STATIC nTotal
PUBLIC cString
nTotal = 7109.50
cString= "Hello"
ON ERROR ? "Variable not available";    && Displays message on error
? nTotal                                && Returns 7109.50
? cString                               && Returns "Hello"
DO Primary                              && Branch to Proc Primary
?
? ***Back to Main.PRG***
? nTotal                                && 7109.50-Still static value after ;
                                         return from lower level procedures

PROCEDURE Primary
?
? ***Proc Primary***                    && Orientation only
STATIC Deadline                         && Var declared Static in Procedure Primary
? Deadline                             && .F. because has no stored value
Deadline = {12/31/99}                   && Initialize variable
? Deadline                             && Now it has a value
DO Proc1                                && Branch to Proc1
RETURN                                  && Return to line after DO Primary in Main

PROCEDURE Proc1
?
? ***Proc Proc1***                      && Orientation only
? cString                               && "Hello" is Public
? nTotal                               && "Var not available"-Static in Main
? Deadline                             && "Var not available"-Static in Primary
DO Sub1                                 && Branch to Sub1
?
? ***Back to Proc1***

```

```

? nTotal          && "Var not available"-Static in another procedure
RETURN           && Return to last line of Primary

PROCEDURE Sub1
?
? "***Proc Sub1**"    && Orientation only
nTotal = 8801.11      && nTotal initialized to new value ;
                     && but still a Static variable
? nTotal            && 8801.11 - declared in current proc
? cString            && "Hello" - Public variable
RETURN             && Return to line after DO Sub1 in Proc1

```

Portability

Not supported in dBASE III PLUS or dBASE IV.

See Also

CLEAR MEMORY, DECLARE, LOCAL, PRIVATE, PUBLIC, RELEASE, STORE

STORE

Memory variables

Stores an expression to specified memory variables or array elements.

Syntax

STORE <exp> TO <memvar list> | <array element list>

or

<memvar> | <array element> = <exp>

<exp> The expression to store.

TO <memvar list> | <array element list> Stores <exp> to the memory variable(s) of <memvar list> or to the array elements of <array element list>. You must DECLARE the array before using STORE.

<memvar> | <array element> = <exp> The single memory variable or array element to store <exp> to with the assignment operator (=). You must DECLARE the array before using =.

Description

Use STORE to store any valid expression to a single memory variable, to several memory variables, or to one or more array elements. Use = to store any valid expression to a single memory variable or a single array element. When <exp> is a field name, the contents of that field are stored. You can specify a field of any data type, including memo.

To specify the *scope* of a variable, use LOCAL, PRIVATE, PUBLIC, or STATIC before assigning a value to the variable. Memory variables are not linked to a particular *session*—only the scope of a variable determines its accessibility within an application.

S

When you issue STORE, dBASE does one of the following:

- Creates a new memory variable and stores *<exp>* to it, or overwrites an existing memory variable with the same name and scope and stores *<exp>* to it. (You can use TYPE() to determine if a memory variable exists). The memory variable is assigned the same data type as *<exp>*. SET SAFETY has no effect on STORE commands.
- Stores *<exp>* to the existing array element you specify. If the data type of the array element is different from that of *<exp>*, STORE changes the array element data type to match the data type of *<exp>*. You can also use AFILL() to store the same value to multiple elements in an array.

If you use STORE to create a variable with the same name as a field in the current table and later use the variable name in a command line, distinguish the memory variable by prefixing it with m->. This is shown in the following example.

```
** Current table has a field named "author"
STORE "Hugo" to author
? author                                && displays contents of field
? m->author                             && displays "Hugo"
```

You can use STORE or STORE MEMO to store a memo field. STORE stores a record's memo field, including carriage return(s) and linefeed(s), to a single memory variable or element of an array. STORE MEMO stores each line of a memo field to one element of an existing array.

Example

The following example uses STORE to place four state abbreviations in four elements of a one-dimensional array named States and then uses a counting DO WHILE loop to retrieve records from the Clients table for the respective states stored in memory:

```
SET TALK OFF
SET SAFETY OFF
DECLARE States[4]
STORE "TX" TO States[1]
STORE "WA" TO States[2]
STORE "GA" TO States[3]
STORE "MN" TO States[4]
USE Clients EXCLUSIVE
INDEX ON Company Tag Company
CLEAR
Cnt = 1
? CENTER("Company Listing for "+States[1] ;
  +", "+States[2]+ ", "+States[3]+ " and " + States[4])
?
DO WHILE Cnt < 5
  SCAN FOR State_Prov = States[Cnt]
  ? Company AT 5, Contact, State_Prov
  ENDSCAN
  Cnt=Cnt+1
ENDDO
CLOSE ALL
SET TALK ON
SET SAFETY ON
```


Portability

Array element lists are not supported in dBASE III PLUS.

See Also

ACCEPT, AFILL(), DECLARE, INPUT, LOCAL, PRIVATE, PUBLIC, RESTORE, SAVE, SET SAFETY, STATIC, STORE MEMO, WAIT

STORE AUTOMEM

Fields and records

Stores the contents of all the current record's fields to a set of memory variables.

Syntax

STORE AUTOMEM

Description

STORE AUTOMEM copies every field of the current record to a set of matching automem variables. Each memory variable has the same name, length, and data type as one of the fields. *Visual* dBASE creates these memory variables if they don't already exist.

Automem variables let you temporarily store the data from table records, manipulate the data as memory variables rather than as field values, and then return the data to the table (using REPLACE AUTOMEM, APPEND AUTOMEM, or INSERT AUTOMEM).

STORE AUTOMEM is one of three commands that create automem variables. The other two, USE <filename> AUTOMEM and CLEAR AUTOMEM, initialize empty automem variables for the fields of the current table. Neither of these commands transfers data to the automem variables it creates, and the variables don't contain data until other commands store it to them. The empty variables created by USE...AUTOMEM and CLEAR AUTOMEM are typically used to append, insert, or replace data from outside the program, such as in a data entry form, into a table. STORE AUTOMEM transfers data from a table into automem variables.

Using memory variables in programming editing sequences gives you more control over data editing than using field values directly. When you specify a field as a GET argument in an @...SAY...GET line, after the field is edited using READ, *Visual* dBASE returns the edited field directly to the table. When you use a memory variable as a GET argument, you can perform other operations on the edited variable, such as validating the data, before using REPLACE AUTOMEM to store the value back in the table.

S**Example**

The following example uses STORE AUTOMEM to copy field values to a set of automem variables for records that are marked for deletion. The LOCATE command initiates a search for the first marked record and a DO WHILE .NOT. EOF() loop with a CONTINUE command copies all other marked records to Temp.DBF:

```
SET SAFETY OFF
SET DELETED OFF
CLOSE DATABASES
```

STORE MEMO

```
USE Clients IN 1
SELECT 1
COPY STRUCTURE TO Temp
USE Temp IN 2
LOCATE FOR DELETED()
IF FOUND()
    DO WHILE .NOT. EOF()
        STORE AUTOMEM
        SELECT 2
        APPEND AUTOMEM
        SELECT 1
        CONTINUE
    ENDDO
    SELECT 2
    BROWSE
ELSE
    ? "No records marked for Deletion"
ENDIF
RETURN
```

Portability

Not supported in dBASE IV or dBASE III PLUS.

See Also

CLEAR AUTOMEM, REPLACE, USE

STORE MEMO

Fields and records

Stores the text of a memo field to an array-type memory variable.

Syntax

STORE MEMO *<memo field>* TO ARRAY *<array name>*

<memo field> The memo field from which text is retrieved and stored in an array.

TO ARRAY <array name> The array in which text from the memo field is stored. Each array element stores one line of text from the memo field. The length of each line can be up to the number of characters specified by SET MEMOWIDTH.

Description

Use STORE MEMO to store the text of memo fields to array memory variables. (STORE AUTOMEM lets you store the contents of the current record in automem variables for all types of fields except memo fields.) After editing the contents of the array elements that hold the contents of a memo field, use REPLACE MEMO to save the new values back in the memo field.

Initialize the array using the DECLARE command before using STORE MEMO.

<array name> must be a one-dimensional array.

Make sure that you initialize arrays used with the STORE MEMO command so that they have enough elements to hold all the memo field text. If the memo field text exceeds the

size of the array, *Visual* dBASE truncates the text to fit the size of the array and the lines at the end of the memo field are lost when you use REPLACE MEMO. Make sure that the array has enough elements to hold the largest memo you plan to edit. You can use MEMLINES() to determine the number of elements you need for any particular memo field you have.

Visual dBASE assigns the character type to each array element that holds memo field text. If the memo field text doesn't fill the array, the remaining array elements retain their original values (.F. as a default).

Example

The following example uses STORE MEMO to place text from a memo field in an array_type memory variable:

```
SET TALK OFF
Colwidth = 50
Mwidth = SET("MEMOWIDTH")
SET MEMOWIDTH TO Colwidth
USE Company
DO WHILE .NOT. EOF()
CLEAR
Linecnt = 1
IF .NOT. ISBLANK(Notes)
  DECLARE Memoline[MEMLINES(Notes)]
  STORE MEMO Notes TO ARRAY Memoline
  DO WHILE Linecnt <= MEMLINES(Notes)
    @ Linecnt,12 GET memoline[linecnt]
    linecnt = linecnt + 1
  ENDDO
  READ
  replace_yn = "C"
  @ linecnt + 2,2 SAY "Do you want to Replace " + ;
    "the original text, Add to it, or Cancel? (R/A/C)";
  GET replace_yn PICTURE "!" VALID replace_yn $ "RAC"
  READ
  DO CASE
    CASE replace_yn = "R"
      REPLACE MEMO Notes WITH ARRAY Memoline
    CASE replace_yn = "A"
      REPLACE MEMO Notes WITH ARRAY Memoline ADDITIVE
    CASE replace_yn = "C"
      CLEAR
      EXIT
  ENDCASE
ENDIF
SKIP
ENDDO
CLOSE ALL
SET MEMOWIDTH TO Mwidth
```

S

Portability

Not supported in dBASE IV or dBASE III PLUS.

See Also

DECLARE, MEMBLINES(), MLINE(), REPLACE MEMO, STORE

STR()**Expressions and type conversion**

Returns the character string equivalent of a specified numeric expression.

Syntax

STR(<expN> [, <length expN> [, <decimals expN> [, <expC>]]])

<expN> The numeric or float expression to return as a character string.

<length expN> The length of the character string to return. The valid range is 1 to 20, inclusive, and includes a decimal point, decimal digits, and minus sign characters. The default is 10. If <length expN> is smaller than the number of integer digits in <expN>, STR() returns asterisks (*).

<decimals expN> The number of characters to reserve for decimal digits. The default and lowest allowable value is 0. If you do not specify a value for <decimals expN>, STR() rounds <expN> to the nearest whole number. If you want to specify a value for <decimals expN>, you must also specify a value for <length expN>.

<expC> The character to pad the beginning of the returned character string with when the length of the returned string is less than <length expN> digits long. The default pad character is a space. If you want to specify a value for <expC>, you must also specify values for <length expN> and <decimals expN>. You can specify more than one character for <expC>, but STR() uses only the first one.

Description

Use STR() to convert numeric data to character data, so you can manipulate it as characters. For example, you can index on a numeric field in combination with a character field by converting the numeric field to character with STR().

dBASE rounds and pads numbers to fit within parameters you set with <length expN> and <decimals expN>, following these rules:

- If <decimals expN> is smaller than the number of decimals in <expN>, STR() rounds to the most accurate number that will fit in <length expN>. For example, STR(10.765,5,1) returns " 10.8" (with a single leading space), and STR(10.765,5,2) returns "10.77".
- If <length expN> isn't large enough for <decimals expN> number of decimal places, STR() rounds <expN> to the most accurate number that will fit in <length expN>. For example, STR(10.765,4,3) returns "10.8".
- If <decimals expN> is larger than the number of decimals in <expN>, and <length expN> is larger than the returned string, STR() adds zeros (0) to the end of the returned string. dBASE only adds enough zero to bring the number of decimal digits to a maximum of <decimals expN>.

- If the returned string is still shorter than *<length expN>*, dBASE pads the left to fill to the length of *<length expN>*. For example, STR(10.765,8,6) returns "10.76500" for a returned length of 8; STR(10.765,7,6) returns "10.7650" for a returned length of 7; and STR(10.765,12,6) returns " 10.765000" (with three leading spaces) for a returned length of 12.

Example

The following example uses STR() to display the contents of a numeric field concatenated with a character string:

```
SET TALK OFF
USE Clients
SET FILTER TO StartBal > 0
* Selects only those records with positive StartBal
GO TOP
? StartBal                                && Returns 456.00
? "The starting balance was $" + ;
  STR(StartBal,8,2)                        && Returns $ 456.00
? "The starting balance can also be formatted as $" + ;
  LTRIM(STR(StartBal,8,2))                 && Returns $456.00
? "Or format the starting balance as " + ;
  LTRIM(STR(StartBal,8,2,"$"))             && Returns $$456.00
```

Portability

The *<exp C>* argument is not supported in dBASE IV or dBASE III PLUS.

See Also

SET POINT, SET SEPARATOR, SUBSTR(), VAL()

STUFF()

String data

Returns a string with specified characters removed and others inserted in their place.

Syntax

STUFF(*<target expC>* | *<target memo field>*,
<start expN>, *<quantity expN>*, *<replacement expC>*)

<target expC> | *<target memo field>* The string or memo field to remove characters from and replace with new characters.

<start expN> The character position in the string or memo field at which to start removing characters.

<quantity expN> The number of characters to remove from the string or memo field.

<replacement expC> The characters to insert in the string or memo field.

Description

STUFF() returns a target character expression or target memo field with a replacement character string inserted at a specified position. Starting at the position you specify,

STUFF()

<start expN>, STUFF() removes a specified number, *<quantity expN>*, of characters from the original string. STUFF() returns a maximum of 32766 characters, the maximum length of a string.

If the target character expression is an empty string or the target memo field is empty, STUFF() returns the replacement string.

If *<start expN>* is less than or equal to 0, STUFF() treats *<start expN>* as 1. If *<quantity expN>* is less than or equal to 0, STUFF() inserts the replacement string at position *<start expN>* without removing any characters from the target.

If *<start expN>* is greater than the length of the target, STUFF() doesn't remove any characters and appends the replacement string to the end of the target.

If the replacement string is empty, STUFF() removes the characters specified by *<quantity expN>* from the target, starting at *<start expN>*, without adding characters.

Example

The following example uses STUFF() to change the text in a series of strings:

```
? STUFF("Jenson",5,1,"e")      && Returns "Jensen"
? STUFF("Johnson",6,0,"t")    && Returns "Johnston"
? STUFF("",2,5,"father")      && Returns "father"
? STUFF("rose",0,1,"n")       && Returns "nose"
? STUFF("rose",5,2,"bud")     && Returns "rosebud"
? STUFF("rosebud",5,3,"")     && Returns "rose"
```

The next example uses STUFF() to replace all occurrences of "&" with "and" in the Company field of Temp.DBF, which is a copy of the Clients table:

```
USE Clients
COPY TO TEMP
USE TEMP
SET TALK OFF
SCAN
  IF "&" $ Company
    REPLACE Company with ;
    STUFF(Company,AT("&",Company),1,"and")
  ENDIF
ENDSCAN
SET TALK ON
RETURN
```

Portability

The *<memo field>* argument isn't supported in dBASE IV or dBASE III PLUS.

See Also

AT(), LEFT(), RAT(), REPLICATE(), RIGHT(), SPACE(), SUBSTR()

SUBSTR()

String data

Returns a substring derived from a specified character string or memo field.

Syntax

SUBSTR(<expC> | <memo field>, <start expN> [, <length expN>])

<expC> | <memo field> The string or memo field to extract characters from.

<start expN> The character position in the string or memo field to start extracting characters.

<length expN> The number of characters to extract from the string or memo field.

Description

Starting in a character expression or memo field at the position you specify for <start expN>, SUBSTR() returns the number of characters you specify for <length expN>. SUBSTR() returns a maximum of 32766 characters, the maximum length of a string. If <length expN> is zero or a negative number, SUBSTR() returns an empty string.

If you don't specify <length expN>, SUBSTR() returns all characters starting from position <start expN> to the end of the string. If <length expN> is greater than the number of characters from <start expN> to the end of the string, SUBSTR() returns only as many characters as are left in the string, without adding space characters to achieve the specified length. You can use LEN() to determine the actual length of the returned string.

The following conditions cause dBASE to return an error:

- <expC> is an empty string
- <memo field> is empty
- <start expN> is zero
- <start expN> is a negative number
- <start expN> is greater than the number of characters in <expC> or <memo field>

When SUBSTR() returns characters from a memo field, it counts two characters for each carriage-return and linefeed combination (CR/LF) in the memo field.

Use the substring operator (\$) to learn if one string exists within another. See Chapter 1 for more information on operators.

Example

The following examples use SUBSTR() to extract a portion of a text string:

? SUBSTR("Data",1)	&& Returns "Data"
? SUBSTR("retrieval",7,3)	&& Returns "val"
? SUBSTR("is",1,1)	&& Returns "i"
? SUBSTR("made",3,1)	&& Returns "d"
? SUBSTR("easy",1)	&& Returns "easy"

SUBSTR() can be used for many ordering and data manipulation tasks. For example, to order a table by the first three numbers of a phone number (disregarding area code)

when data is entered in a character field in the format "451-463-9000", create an index tag using SUBSTR().

```
USE COMPANY EXCLUSIVE
INDEX ON SUBSTR(Phone,5,3) TAG Phexch
```

To convert field data entered as uppercase to upper- and lowercase, use the following commands (field length = 20).

```
USE Address
REPLACE ALL Lname WITH UPPER(SUBSTR(Lname,1,1))+LOWER(SUBSTR(Lname,2,19))
```

The previous example applies only to cases where a single name or text string resides in a field. If you want to capitalize all first letters of multiple strings in a field, use PROPER():

```
REPLACE ALL Lname with PROPER(Lname)
```

Portability

The *<memo field>* argument isn't supported in dBASE III PLUS. Both dBASE IV and dBASE III PLUS limit the return value of SUBSTR() to 254 characters, and both return an error if *<start expN>* or *<length expN>* is zero or if *<length expN>* is a negative number.

See Also

AT(), LEFT(), LEN(), PROPER(), RAT(), RIGHT(), STUFF()

SUM

Table organization

Computes a total for specified numeric and float fields in the current table and stores the results in memory variables or an array.

Syntax

```
SUM
[<exp list>]
[<scope>]
[FOR <condition 1>]
[WHILE <condition 2>]
[TO <memvar list> | TO ARRAY <array name>]
```

<exp list> The numeric or float fields, expressions incorporating numeric or float fields, or expressions converting character fields to numeric values to sum.

<scope> The number of records to sum. RECORD <n> identifies a single record by its record number. NEXT <n> identifies n records, beginning with the current record. ALL specifies all records. REST specifies all records from the current record to the end of the file.

FOR <condition 1>

WHILE <condition 2> Determines which records are affected by SUM. FOR restricts SUM to records that meet <condition 1>. WHILE starts processing with the current record and continues with each subsequent record as long as <condition 2> is true.

TO <memvar list> | TO ARRAY <array name> TO <memvar list> specifies a list of memory variables where you want to store totals. If <memvar list> includes array subscripts, the array(s) must already exist. TO ARRAY <array name> specifies an array in which you want to store totals.

Description

The SUM command totals the value of numeric expressions and stores the results in a list of memory variables or individual elements of an array. If SET TALK is ON, DBASE also displays results in the results pane of the Command window.

The number of memory variables specified must be exactly the same as the number of specified numeric expressions you want to total. The position of fields in <exp list> determines the order of values passed to <memvar list>. If you specify an array, it must be one-dimensional, and you must initialize individual elements before storing results in the array using the SUM command.

SUM is similar to TOTAL, which operates on an indexed or sorted table to create a second table containing the sums of the numeric and float fields of records grouped on a key expression.

Example

The following example uses SUM to calculate the total of year-to-date sales of all companies:

```
USE Company
SUM Ytd_sales TO Ytd_sum
? "The total Ytd Sale was $", ;
Ytd_sum PICTURE "99,999,999.99"
```

In this example there is no need to index the table, since every record must be read.

See Also

AVERAGE, CALCULATE, COUNT, TOTAL

SUSPEND

Error handling and debugging

Suspends program execution, temporarily passing control to the Command window.

Syntax

SUSPEND

Description

SUSPEND lets you interrupt program execution at a specific point, a *break point*. The program remains suspended until you issue RESUME or CANCEL, or until you exit dBASE. If you issue RESUME, the program resumes from the break point. If you issue CANCEL, dBASE cancels program execution and clears it from memory. (CANCEL

S

cancels all files called with DO, including ones that are suspended. You must close any procedure files with CLOSE PROCEDURE.)

While a program is suspended, you can enter commands in the Command window. For example, you can check and change the status of files, memory variables, SET commands, and so on; however, dBASE ignores any changes you make to the program while it is suspended. If you want to correct a suspended program, issue CANCEL, edit the program, and then run it again.

If you initialize memory variables in the Command window while a program is suspended, dBASE makes them private at the program level that suspension occurred.

You should not return to a suspended program by issuing DO <filename> in the Command window. If you do so, you might eventually run out of memory. You will also end up with "nested" SUSPEND statements, and may not know that a program is still suspended. If you want to run a suspended program from the beginning, issue CANCEL and then DO <filename>.

Example

The following program prompts the entry of a 2-letter state abbreviation and lists the clients within that state. If the program fails to return a list of clients, the programmer might insert the SUSPEND command just after the second CLEAR to halt the program so that trouble shooting commands could be issued at the Command window such as: ? mState to determine the value in the variable mState, LIST FOR STATE_PROV = "CA", DISPLAY MEMORY, DISPLAY STATUS. Issue the command RESUME when ready to proceed with the remainder of the program:

```
CLEAR
SET TALK OFF
USE CLIENTS
ACCEPT "Enter 2 letter State abbreviation: " TO mState
CLEAR
SUSPEND                                && To be removed after troubleshooting
? CENTER("Clients in "+UPPER(mState))
?
SCAN FOR State_Prov = UPPER(mState)
    ? Company, Contact, Startbal
ENDSCAN
RETURN
```

See Also

CANCEL, DO, RESUME, QUIT

TAG()

Table organization

Returns the name of an .NDX file or .MDX file tag name.

Syntax

TAG([<.mdx filename expC>] <index number expN> [,<alias>])

<.mdx filename expC> The name of the multiple index file in which to look for a tag in the <index number expN> position.

<index number expN> The number of the open .NDX file or index tag in the multiple index file whose name you want to return.

<alias> A work area number (1 through 225), letter (A through J), or alias name. The work area letter or alias name must be enclosed in quotes.

Description

TAG() returns the name of an .NDX file or .MDX tag for the index specified by <index number expN>. The index number indicates the position of an index in the list of open indexes in the current or specified work area opened with the USE or SET INDEX commands. If no index number is specified, TAG() returns the name of the master index.

If you specify an .MDX file name, TAG() returns tag names that appear in the specified multiple index file. The order in which tag names are listed in the multiple index file determines the order in which TAG() returns tag names.

If you don't specify an .MDX file name, TAG() returns the tag name in the open index list and checks .NDX files first. TAG() next checks the production .MDX index tags and then other open .MDX files in the order you opened them.

If no index or tag exists in the specified position, TAG() returns an empty string ("").

Example

The following example uses TAG() to determine the names of index files open for the current table:

```
USE Company EXCLUSIVE
INDEX ON CompCode TAG CompCode
INDEX ON Company TAG Company
INDEX ON Zip_P_Code TAG Zip
* There are now at least 3 indexes in Company.mdx.
FOR i=1 TO TAGCOUNT()
  * TAGCOUNT() is the total number of indexes in Company.mdx
  ? "Tag",i, TAG(i)
NEXT i
```

T

Portability

Not supported in dBASE III PLUS.

See Also

DBF(), DISPLAY STATUS, KEY(), MDX(), NDX(), ORDER(), SET INDEX, SET ORDER, TAGCOUNT(), TAGNO(), USE

TAGCOUNT()**Table organization**

Returns the number of active indexes in a specified work area or .MDX multiple index file.

Syntax

TAGCOUNT([<.mdx filename> [,<alias>]])

<.mdx filename> Specifies the multiple index file that contains the index tag you want to check. If you specify a file without including its path, *Visual* dBASE looks for the file in the current directory, then in the path you specify with SET PATH. If you specify a file without including its extension, *Visual* dBASE assumes an .MDX extension.

<alias> A work area number (1 through 225), letter (A through J), or alias name. The work area letter or alias name must be enclosed in quotes.

Description

TAGCOUNT() returns the total number of open indexes or the number of index tag names in a specified .MDX file. TAGCOUNT() returns 0 if there are no indexes or index tags open for the current or specified work area, or if the multiple index file specified with <filename> does not exist. If you do not specify an .MDX file name, TAGCOUNT() returns the total number of indexes in the specified work area (.NDX files are included). If you do not specify an alias, TAGCOUNT() returns the total number of indexes in the current work area.

Example

The following example uses TAGCOUNT() to determine the number of indexes open for the current table:

```
CLOSE ALL
USE Company IN SELECT() EXCLUSIVE
SELECT Company
INDEX ON CompCode TAG CompCode
INDEX ON Company TAG Company
* Compcode and Company are in the production mdx
INDEX ON Zip_P_Code TAG Zip OF Location
* Zip is in Location.mdx
INDEX ON City TO City
* City is in City.ndx
SET INDEX TO CITY, Location ORDER Zip
* There are now at least 2 tags in Company.mdx
* There is one in Location.mdx
* There is another index open in City.ndx
? "There are " + LTRIM(STR(TAGCOUNT())) + ;
  " active indexes in workarea " + ;
  LTRIM(STR(WORKAREA()))
```

WAIT
DISPLAY STATUS

Portability

Not supported in dBASE III PLUS.

See Also

DBF(), DISPLAY STATUS, KEY(), MDX(), NDX(), ORDER(), SET INDEX, SET ORDER, TAG(), TAGNO(), USE, WORKAREA()

TAGNO()

Table organization

Returns the index number of the specified index.

Syntax

TAGNO([<tag name expC> [,<.mdx filename> [,<alias>]]])

<tag name expC> The name of the index tag that you want to return the position of. If you don't specify a tag name, TAGNO() returns the position of the master index.

<.mdx filename> The name of the multiple index file that contains the specified index tag. If you don't specify an .MDX file name, TAGNO() returns the position of the tag name for all open index files in the same work area, including .NDX files at the top of the index list.

<alias> A work area number (1 through 225), letter (A through J), or alias name. The work area letter or alias name must be enclosed in quotes.

Description

TAGNO() returns a number that indicates the position of the specified index name in the list of open indexes in the current or specified work area. The order of indexes is determined by the order in which they were opened with the USE or SET INDEX commands.

If you don't specify a tag name, TAGNO() returns the number of the master index. If you don't specify an .MDX file name, TAGNO() searches the list of open index files in the specified work area, including .NDX files. If you don't specify an alias, TAGNO() operates on the list of open indexes in the current work area.

TAGNO() returns an error if the specified index tag or .MDX file does not exist.

Example

The following example uses TAGNO() to determine the number of the specified index (.NDX) files:

```
USE Company EXCLUSIVE
INDEX ON CompCode TO CompCode
INDEX ON Company TO Company
INDEX ON Zip_Postal TO Zip
INDEX ON State_prov TO State_Prov
```

T

TAN()

```
INDEX ON City          TO City
SET INDEX TO CompCode, Company, City, State_Prov, Zip
? TAG( ), TAGNO( )
* eg "COMPCODE"      8
? TAGNO("CompCode")      && e.g. 8
? TAGNO("Company")      && e.g. 1
? TAGNO("City")      && e.g. 6
? TAGNO("State")      && e.g. 3
? TAGNO("Zip")      && e.g. 11
```

Portability

Not supported in dBASE III PLUS.

See Also

DBF(), DISPLAY STATUS, KEY(), MDX(), NDX(), ORDER(), SET INDEX, SET ORDER, TAG(), TAGCOUNT(), USE, WORKAREA()

TAN()

Numeric data

Returns the trigonometric tangent of an angle.

Syntax

TAN(<expN>)

<expN> The size of the angle in radians. To convert an angle's degree value to radians, use DTOR(). For example, to find the tangent of a 30-degree angle, use TAN(DTOR(30)).

Description

TAN() calculates the ratio between the side opposite an angle and the side adjacent to the angle in a right triangle. TAN() returns a float that increases from zero to plus or minus infinity. TAN() returns zero when <expN> is 0, pi, or 2*pi radians. TAN() is undefined (returns infinity) when <expN> is pi/2 or 3*pi/2 radians.

Use SET DECIMALS to set the number of decimal places TAN() displays.

The cotangent of an angle is the reciprocal of the tangent of the angle. To return the cotangent of an angle, use 1/TAN().

Example

The following examples use TAN() to return the tangent (in radians) of a defined angle:

```
SET DECIMALS TO 6
? TAN(PI())      && Returns 0.000000
? TAN(PI()/2)    && Returns infinity
? TAN(PI()/4)    && Returns 1.000000
? TAN(DTOR(150)) && Returns -0.577350
? TAN(DTOR(-150)) && Returns 0.577350
```

Portability

Not supported in dBASE III PLUS.

See Also

ATAN(), ATN2(), COS(), DTOR(), PI(), RTOD(), SET DECIMALS, SIN()

TARGET()

Table organization

Returns the name of a table linked to the current or specified work area.

Syntax

TARGET(<expN> [, <alias>])

<expN> Specifies the position of the relation in the SET RELATION list for the current or specified table that you want to determine the relation for.

<alias> Specifies the work area from which you define a relation with the SET RELATION command. If you don't specify an alias for the work area, TARGET() assumes that the relation was set from the current work area. You can enter a work area number (1 through 225), letter (A through J), or alias name. The work area letter or alias name must be enclosed in quotes.

Description

TARGET() returns the name of a table that is linked to the table in the current or specified work area by relations defined with the SET RELATION command. If you do not specify an alias, TARGET() assumes the relation is set from the current work area. The TARGET() function returns an empty string ("") if no relation is set in the <expN> position of the SET RELATION list.

Example

The following example uses TARGET() to determine the names of the child tables linked to Company in the following sample:

```
CLOSE DATABASE
USE Contact EXCLUSIVE
INDEX ON CompCode TAG CompCode
SELECT 2
USE Typeco EXCLUSIVE
SELECT Typeco
INDEX ON Type TAG Type
SELECT 3
USE Company
SET RELATION TO CompCode INTO Contact
SET RELATION TO Type INTO Typeco ADDITIVE
? "RELATION:",RELATION(1) , "TARGET:",TARGET(1)
* displays Compcode CONTACT
? "RELATION:",RELATION(2) , "TARGET:",TARGET(2)
* displays Compcode TYPECO
select 20
```

TEXT

```
? "TARGET:",TARGET(1,"Company")
* displays CONTACT
```

&& works from any work area

Portability

Not supported in dBASE IV or dBASE III PLUS.

See Also

CREATE, CREATE VIEW...FROM ENVIRONMENT, DISPLAY STATUS, RELATION(), SET(), SET RELATION, SET VIEW

TEXT

Input/Output

Displays the lines between TEXT and ENDTEXT as a block of text. This command is supported primarily for compatibility with dBASE IV. In *Visual* dBASE, use DEFINE with the Text class to display text in forms.

For complete syntax information on TEXT, see online Help. For information about working with forms, see the Forms chapters in the *User's Guide*.

TIME()

Date and time data

Returns the system time as a character string in HH:MM:SS or HH:MM:SS.hh format.

Syntax

TIME([<exp>])

<exp> Any expression, which causes TIME() to return the current time to the hundredth of a second.

Description

TIME() returns a character expression that is your system time. If you don't pass TIME() an expression, it returns the current system time in HH:MM:SS format, where HH is the hour, MM the minutes, and SS the seconds.

If you pass TIME() an expression, it returns the current system time in HH:MM:SS.hh, where .hh is hundredths of a second. The expression value you pass to TIME() has no effect on the time it returns other than to make it include hundredths of a second.

Example

See the example of SECONDS() for an example of TIME().

See Also

ELAPSED(), SET TIME

TOTAL

Table organization

Creates a table that stores totals for specified numeric and float fields of records grouped by common key values.

Syntax

```
TOTAL ON <key expC> TO <filename> | ? | <filename skeleton>
[[TYPE] PARADOX | DBASE]
[<scope>]
[FOR <condition 1>]
[WHILE <condition 2>]
[FIELDS <field list>]
```

<key expC> The key expression of the master index or the name of the field on which the current table has been sorted.

TO <filename> | ? | <filename skeleton> Directs output to the dBASE table named <filename>. By default, *Visual* dBASE assigns a .DBF extension to <filename> and saves the file in the current directory. The ? and <filename skeleton> options display a dialog box, in which you specify the name of the target file and the directory to save it in.

You can also create a table in a database (defined using the BDE Configuration Utility) by specifying the database as a prefix (enclosed in colons) to the name of the table, that is, *:database name:table name*. If the database is not already open, *Visual* dBASE displays a dialog box, in which you specify the parameters, such as a login name and password, necessary to establish a connection to that database.

[TYPE] PARADOX | DBASE Specifies the type of table to create. The TYPE keyword is included for readability only; it has no effect on the operation of the command.

Specifying PARADOX creates a Paradox table with a .DB extension.

Specifying DBASE creates a dBASE table (the default). If you don't include an extension for <filename>, dBASE assigns a .DBF extension.

<scope> The number of records to total. RECORD <n> identifies a single record by its record number. NEXT <n> identifies *n* records, beginning with the current record. ALL specifies all records. REST specifies all records from the current record to the end of the file.

FOR <condition 1>

WHILE <condition 2> Determines which records are affected by TOTAL. FOR restricts TOTAL to records that meet <condition 1>. WHILE starts processing with the current record and continues with each subsequent record as long as <condition 2> is true.

FIELDS <field list> Specifies which numeric and float fields to total. If you don't include FIELDS, dBASE totals all numeric and float fields.

Description

Use TOTAL to total the value of numeric fields in a table and create a second table to store the results. The numeric fields in the table storing the results contain totals for all records that have the same key value in the original table.

T

The current table must be either indexed or sorted on the key field. All records with the same key field become a single record in the table storing the result totals. All numeric fields appearing in the fields list contain totals. All other fields contain data from the first record of the set of records with identical keys.

TOTAL is similar to SUM, except that SUM operates on an indexed or unindexed table, returning a sum for all records of each numeric field. SUM doesn't create another table, but stores the results to memory variables or an array.

Example

The following example uses TOTAL on the Company table to calculate the total of year to date sales in each state:

```
CLOSE DATABASE
USE Company EXCLUSIVE
INDEX ON State_Prov TAG State
* First index Company table by State
* Index requires exclusive on
USE COMPANY SHARED
* TOTAL does not need exclusive on
TOTAL ON State_Prov TO StateTot
SELECT 2
USE StateTot
BROWSE FIELDS State_Prov, Ytd_sales TITLE "Total sales per state"
```

StateTot contains one record per state.

See Also

AVERAGE, CALCULATE, COUNT, SUM

TRANSFORM()

String data

Returns a string containing data in a specified format.

Syntax

TRANSFORM(<exp>, <picture expC>)

<exp> The character, numeric, logical, or date expression to be formatted.

<picture expC> The string containing the template characters necessary to format <exp>. The template characters are the same as those used in the PICTURE or FUNCTION option of ? and ??, the Function property of the Entryfield class, etc. Precede any FUNCTION template characters used with TRANSFORM() with an explicit @ symbol. For more information about the FUNCTION option, see Function in Chapter 8.

Description

TRANSFORM() returns an expression in the PICTURE or FUNCTION format you indicate with <picture expC>. Use TRANSFORM() to format data when the command you're using allows expressions but doesn't include a PICTURE or FUNCTION option. For example, use TRANSFORM() to format data in report and label forms, and to

format DISPLAY and LIST output. Such formatting includes aligning text and displaying numbers in scientific notation.

Example

The following example uses TRANSFORM() to print or display Startbal amounts with commas and Baldate in English format (day/month/year):

```
CLEAR
USE Clients EXCLUSIVE
INDEX ON Company TO Comp
SET FIELDS TO Company, Startbal, Baldate
SCAN
  ? Company, TRANSFORM(Startbal,"999,999.99"), TRANSFORM(Baldate,"@E")
  * Display balance with commas and date in English format
ENDSCAN
CLOSE DATABASES
```

See Also

?, ??, CLASS ENTRYFIELD

TRIM()

String data

Returns a string with no trailing space characters.

Syntax

TRIM(<expC> | <memo field>)

<expC> | <memo field> The string or memo field to remove the trailing space characters from.

Description

TRIM() returns a character expression or memo field with no trailing space characters. TRIM() returns a maximum of 32766 characters, the maximum length of a string. TRIM() is identical to RTRIM().

Using TRIM() with a memo field removes trailing spaces only at the end of the last line of the field. To remove trailing spaces from a particular line of a memo field, use LTRIM() with MLINE().

To remove *leading* space characters from a string or memo field, use LTRIM().

Example

The following example uses TRIM() to remove trailing spaces from text in a character field:

```
USE Company
X = City + State + zip
? X
* Returns "Scotts Valley      CA      95066"
X=TRIM(City)+"", "+TRIM(State)+SPACE(2)+ZIP
```

T

? X
 * Returns "Scotts Valley, CA 95066"

Portability

The *<memo field>* argument isn't supported in dBASE IV or dBASE III PLUS. Both dBASE IV and dBASE III PLUS limit the return value of TRIM() to 254 characters.

See Also

LEFT(), LTRIM(), MLINE(), RIGHT(), STR(), SUBSTR()

TYPE

Disk and file utilities

Display the contents of an ASCII file.

Syntax

```
TYPE <filename 1> | ? | <filename skeleton 1>
[MORE]
[NUMBER]
[TO FILE <filename 2> | ? | <filename skeleton 2>] | [TO PRINTER]
```

<filename> | ? | <filename skeleton> The file whose contents to display, also called the source file. TYPE ? and TYPE <filename skeleton> display a dialog box from which you can select a file. If you specify a file without including its path, dBASE looks for the file in the current directory, then in the path you specify with SET PATH. You must specify a file-name extension.

MORE Pauses output when it fills the Command window; otherwise, the output scrolls through the Command window to the end of the file.

NUMBER Precedes each line of output with its line number.

TO FILE <filename 2> | ? | <filename skeleton> Directs output to the text file <filename 2>, also called the target file, as well as to the results pane of the Command window. By default, dBASE assigns a .TXT extension to <filename 2> and saves the file in the current directory. The ? and <filename skeleton> options display a dialog box in which you specify the name of the target file and the directory to save it in.

TO PRINTER Directs output to the printer, as well as to the results pane of the Command window.

Description

Use TYPE to display, copy, or print the contents of ASCII files.

If you TYPE a file TO FILE or TO PRINTER, dBASE adds two lines of output at the beginning of the saved or printed output. The first line is a blank line, and the second line contains the full path name and date stamp of the source file. If you specify NUMBER, these two lines are not numbered; numbering begins with 1 at the first actual line of the source file. If you specify MORE and cancel output before completion, *** INTERRUPTED *** appears in the results pane of the Command window, but does not appear in the incomplete saved or printed output.

If SET SAFETY is ON and a file exists with the same name as the target file, dBASE displays a dialog box asking if you want to overwrite the file. If SET SAFETY is OFF, any existing file with the same name is overwritten without warning.

Example

The following examples use TYPE:

TYPE Results.txt	&& Writes to the screen
TYPE Results.txt MORE	&& Pauses at each screenful
TYPE Myfile.prg NUMBER	&& Displays line numbers
TYPE Myfile.prg NUMBER TO FILE Myfile1.prg	&& Writes with numbers to new file
TYPE Myfile.prg NUMBER TO FILE ?	&& Opens dialog box for file name
TYPE Myfile.prg TO PRINTER	&& Prints Myfile.prg

Portability

Only a TO PRINT option is supported in dBASE III PLUS. The MORE, ?, and <filename skeleton> options are not supported in dBASE IV. If you don't specify a file extension with the TO FILE <filename> option, dBASE IV adds .PRT instead of .TXT.

See Also

COPY FILE, EJECT, SET ALTERNATE, SET PRINTER, SET SAFETY

TYPE()

Expressions and type conversion

Returns a character string that indicates a specified expression's data type.

Syntax

TYPE(<exp> | <expC>)

<exp> | <expC> The expression whose type to evaluate and return.

Description

TYPE() can evaluate only character-based expressions. You can use <exp> (no quotes) only if you are evaluating a memory variable that contains either the name of a database field or character string that references an expression. Otherwise, use <expC> to evaluate a literal expression, a field name, or a memory variable.

For example, if your database has a field named "START," you can determine the type of this field in two ways. Either use TYPE("START"), or store the field name to a memory variable (STORE "START" TO mvar) and use TYPE(mvar).

A similar principle applies to memory variables that are not field names:

- If you issue STORE "1+2=5" TO mvar, TYPE(mvar) returns L. This is because dBASE interprets TYPE(mvar) as "What type of expression does the character string contained in mvar represent?" The character string contained in mvar is "1+2=5", which is a logical expression, so dBASE returns L.

T

TYPE()

- If you issue STORE "1+2=5" TO mvar, TYPE("mvar") returns C. However, dBASE interprets TYPE("mvar") as "What type of data does the variable mvar contain?" Since "1+2=5" is a character string, dBASE returns C in this case.

Memory variables may also evaluate to other memory variables. For example, if you issue STORE 10 TO mnum and STORE "mnum" TO mvar, TYPE(mvar) returns N. Again, this is because dBASE evaluates TYPE(mvar) as "What type of expression does the character string contained in mvar represent?" The character string in mvar is "mnum," which represents a variable that contains the number 10, so dBASE returns N.

If you want to evaluate a memory variable that does not evaluate to a character string, you must use <expC>, as shown in the following example.

```
STORE DATE() TO mvar
? TYPE(mvar)                && returns an error because mvar does ;
                             not contain a character string.
? TYPE("mvar")              && returns D
```

TYPE() always returns a character string. The following table lists the values TYPE() returns.

If <exp> <expC>contains this type of data	TYPE() returns
dBASE array variable	A
dBASE or Paradox binary field (BLOB)	B
dBASE bookmark variable	BM
dBASE character field or string variable, Paradox alphanumeric field	C
dBASE code block variable	CB
dBASE date field or variable, Paradox date field	D
dBASE float field, Paradox numeric or currency field	F
dBASE function pointer variable	FP
OLE (general)	G
dBASE logical field or variable	L
dBASE or Paradox memo field	M
dBASE numeric field or integer variable	N
dBASE object variable	O
dBASE SAVE SCREEN variable	S
dBASE or Paradox undefined variable, field, or invalid expression	U

Example

The following examples use TYPE() to determine the data type of a specified expression.

```
USE Clients
? TYPE("Company")          && Returns C
? TYPE("BalDate")          && Returns D
? TYPE("StartBal")         && Returns N
mDate={12/31/99}           && date type variable
? TYPE("mdate")            && Returns D
? TYPE(mdate)              && Returns error message
mField = "Startbal"        && name of a field
? TYPE(mfield)             && Returns N
```

```

mStartbal=Startbal      && contents of a field
? TYPE(mStartbal)       && Returns error message
? TYPE("mStartbal")     && Returns F

```

See Also

EMPTY(), ISBLANK(), STORE

UNIQUE()

Table organization

Indicates whether a specified index was created with the UNIQUE keyword (or with SET UNIQUE ON).

Syntax

```
UNIQUE([<.mdx filename>] <index position expN> [, <alias>])
```

<.mdx filename> Specifies a multiple index file that contains the index tag you want to check.

<index position expN> Selects an index file or tag by the position of an index tag in an .MDX file or the position of an index file in the list of open indexes for the current or a specified table. The index position number specifies the position of the index within the list of open indexes as opened with the SET INDEX or USE commands.

<alias> A work area number (1 through 225), letter (A through J), or alias name. The work area letter or alias name must be enclosed in quotes.

Description

The UNIQUE() function returns .T. if the index specified by the optional *<index position expN>* parameter was created with the INDEX...UNIQUE option, or INDEX with SET UNIQUE ON. If you do not specify an index number, UNIQUE() checks the master index or index tag for the current or specified work area.

The UNIQUE() function returns .F. when

- The master index or the specified index or index tag number was not created with the INDEX...UNIQUE option or INDEX with UNIQUE set ON.
- You do not include an index number and the current table does not have a master index.
- No open index or index tag in the current or specified work area has the specified index number.
- UNIQUE() returns an error if a specified index or filename does not exist.

Example

The following example uses UNIQUE() to determine if the key for a specified index was created as a UNIQUE index:

```

USE Company EXCLUSIVE
INDEX ON CompCode TAG CompCode
SET ORDER TO TAG Compcode

```

```
? TAG(), "Unique = ", UNIQUE()
INDEX ON State_Prov TAG State OF Location UNIQUE
SET ORDER TO TAG State OF Location
? TAG(), "Unique = ", UNIQUE()
? TAG(1), "Unique = ", UNIQUE(1), MDX(1)
* UNIQUE can reference an index by position
* This is the first index of Company.mdx
```

Portability

Not supported in dBASE III PLUS.

See Also

DESCENDING(), FOR(), INDEX, KEY(), MDX(), NDX(), ORDER(), SET UNIQUE, TAG(), TAGCOUNT(), TAGNO(), WORKAREA()

UNLOCK

Shared data

Unlocks the current table if you locked it with FLOCK(). Unlocks all records in the current table you locked with RLOCK() or LOCK().

Syntax

```
UNLOCK
[ALL | IN <alias>]
```

ALL In all work areas, unlocks all tables you locked with FLOCK() and all records you locked with RLOCK() or LOCK().

IN <alias> Unlocks the alias table <alias> if you locked it with FLOCK(), or unlocks all of its records you locked with RLOCK() or LOCK(). <alias> is a work area number (1 through 225), letter (A through J), or alias name. The work area letter or alias name must be enclosed in quotes. If you don't include <alias>, UNLOCK unlocks the current table.

Description

Use UNLOCK to unlock file locks you obtained with FLOCK(), or to unlock record locks you obtained with RLOCK() or LOCK(). Issue UNLOCK at the same workstation as the one at which you issued the FLOCK(), RLOCK(), and LOCK() functions. UNLOCK can't release locks obtained through other workstations.

If you're browsing or editing a table, and the record pointer is on a record or an entry field representing a field in the table, pressing *Ctrl+O* toggles the current record between a locked and unlocked state. That is, if the record is locked, *Ctrl+O* unlocks it; if the record is unlocked, *Ctrl+O* locks it.

When you set a relation to a parent table with SET RELATION and then unlock the parent table or records in the parent table with UNLOCK, dBASE also unlocks related tables or records. For more information on relating tables, see SET RELATION.

Example

See FLOCK() for an example for UNLOCK.

Portability

Not supported in dBASE III PLUS.

See Also

FLOCK(), RLOCK(), SET RELATION

UPDATE

Table organization

Replaces data in the specified fields of the current table with data from another table.

Syntax

```
UPDATE ON <key exp> FROM <alias>
REPLACE <field 1> WITH <exp 1>
    [, <field 2> WITH <exp 2>...]
[RANDOM]
[REINDEX]
```

<key exp> The key expression that is common to both the current table and a table in a second work area.

FROM <alias> Specifies a table open in another work area that provides updates to the current table.

REPLACE <field 1> WITH <exp 1> Specifies the field in the current table to be updated with the table specified by FROM <alias>. If a field specified by WITH <exp 1> is in a different work area from the current table, identify the field by its alias, for example, *alias->field*.

[, <field n> WITH <exp n> ...] Specifies additional fields to be updated.

RANDOM Specified when the updating table is neither indexed nor sorted. (Current table must be indexed on the key expression common to both tables.)

REINDEX Specifies that all affected non-master indexes are rebuilt once the update operation finishes.

Description

The UPDATE command uses data from a specified table to replace field values in the current table. It makes the changes by matching records in the two files based on a single key field.

The current table must be indexed on the field in the key expression. Unless the RANDOM option is used, the table in the specified work area should also be indexed or sorted on the same field. Fields that are included in the key expression must have identical names in the two tables.

U-V

UPDATED()

Example

The following example uses UPDATE to recalculate the totals for each order in the Orders table from the LineItem table. UPDATE cannot look up multiple records, so LineItem is totalled on Order_No to a new table, LineTot, and Orders is updated from LineTot:

```
SET TALK ON
* to show the record counts
CLOSE ALL
USE ORDERS EXCLUSIVE
SELECT 2
USE LineItem
INDEX ON Order_no TAG Orders
TOTAL ON Order_no TO LineTot
USE LineTot
SELECT Orders
UPDATE ON Order_no FROM LineTot REPLACE Tot_inv WITH LineTot->Total
* Only orders with lineitems will be updated. With Talk ON you will see
* the number of records that were updated.
```

See Also

APPEND FROM, JOIN, REPLACE, SELECT, SET RELATION

UPDATED()

Input/Output

Returns .T. (true) if you changed the contents of any @...GET fields or memory variables in the results pane of the Command window or the current dBASE IV window. This command is supported primarily for use with dBASE IV windows. In *Visual dBASE*, use properties such as OnChange to manage changed information in forms.

For complete syntax information on UPDATED(), see online Help. For information about working with forms, see the Forms chapters in the *User's Guide*.

UPPER()

String data

Converts all lowercase characters in a string to uppercase and returns the resulting string.

Syntax

UPPER(<expC> | <memo field>)

<expC> | <memo field> The character string or memo field to convert to uppercase.

Description

UPPER() converts the lowercase alphabetic characters in a character expression or memo field to uppercase. UPPER() ignores digits and other characters. UPPER() returns a maximum of 32766 characters, the maximum length of a string.

The current language driver defines the character values that are lowercase and uppercase alphabetic. In a U.S. language driver, a lowercase alphabetic character is from a to z, and an uppercase alphabetic character is from A to Z. See Appendix C in the *Programmer's Guide* for information about language drivers.

Example

The following example uses UPPER() to convert lowercase text to uppercase when comparing two strings:

```
? UPPER("Technical")           && Returns "TECHNICAL"
? UPPER("Technical") = "Technical";  && Returns .F.
? UPPER("")                     && Returns ""
? UPPER("12 apples")            && Returns "12 APPLES"
```

UPPER() is frequently used to ensure that character data matches for various comparisons. The following example uses UPPER() to ensure a match during a SEEK:

```
SET EXACT OFF
USE Animals EXCLUSIVE
INDEX ON UPPER(Name) TAG Name
mSearch="Boa"
SEEK UPPER(msearch)
IF FOUND()
    EDIT
ENDIF
RETURN
```

Portability

The *<memo field>* argument isn't supported in dBASE IV or dBASE III PLUS.

See Also

CHARSET(), ISALPHA(), ISLOWER(), ISUPPER(), LDRIVER(), LOWER(), PROPER(), SET LDCHECK

USE

Table basics

Opens the specified table, associated memo (.DBT) files, and production index (.MDX) files, if any.

Syntax

```
USE
[<filename 1> | ? | <filename skeleton 1>
[[TYPE] PARADOX | DBASE]
[IN <alias>]
[INDEX <index name list> |
  <? list> | <index name skeleton list>]
[ORDER [TAG] <.ndx filename> |
  <tag name> [OF <.mdx filename>]]
[AGAIN]
[ALIAS <alias name>]
```

[AUTOMEM]
 [EXCLUSIVE | SHARED]
 [NOSAVE]
 [NOUPDATE]]

<filename 1> | ? | <filename skeleton 1> The table file you want to open. USE ? and USE <filename skeleton 1> displays a dialog box, from which you can select a table file. If you specify a file without including its path, *Visual dBASE* looks for the file in the current directory, then in the path you specify with SET PATH. If you specify a file without including an extension or specifying its type, *Visual dBASE* assumes the file type specified with the SET DBTYPE command.

You can also open a table in a database (defined using the BDE Configuration Utility) by specifying the database as a prefix (enclosed in colons) to the name of the table, that is, :database name:table name. If the database is not already open, *Visual dBASE* displays a dialog box in which you specify the parameters, such as a login name and password, necessary to establish a connection to that database.

[[TYPE] PARADOX | DBASE] Specifies the type of table to open. The TYPE keyword is included for readability only; it has no effect on the operation of the command.

Specifying PARADOX opens a Paradox table with a .DB extension.

Specifying DBASE opens a dBASE table (the default). If you don't include an extension for <filename>, *Visual dBASE* assumes a .DBF extension.

IN <alias> Specifies a work area. You can enter a work area number (1 through 225), letter (A through J), or alias name. The work area letter or alias name must be enclosed in quotes.

INDEX <.index name list> | <? list> | <index name skeleton list> Applicable to dBASE indexes only. (Indexes on Paradox and SQL tables are specified by the ORDER clause.) Opens up to 100 individual index files for the specified table. <index name list> can include single (.NDX) and multiple index file (.MDX) names. If the first file in <index name list> is a single index file, and you don't use the ORDER option, the single index file becomes the master index. INDEX <? list> and INDEX <index name skeleton list> displays a dialog box, from which you can choose an existing index file.

ORDER [TAG] <tag name> Makes the <tag name> index file the master index. When opening a Paradox table, you may specify a secondary index as the master index; otherwise, the PRIMARY index is used. When opening an SQL table, you can also specify an index to use as the master index.

If you don't include the ORDER clause and the first file name after INDEX is a single index .NDX file, the single index file is the master index. If you don't include ORDER and the first file name after INDEX is a multiple index .MDX file, the table doesn't use a master index.

OF <.ndx filename> The multiple index file that includes <tag name>. Without OF <filename>, dBASE searches for <tag name> in a multiple index file with the same root name as the table.

ORDER [TAG] <.ndx filename> Makes the single index file, <.ndx filename>, the master index.

AGAIN Opens a table and its related index files in the current or specified work area, leaving the table open in one or more other work areas.

ALIAS <alias name> Specifies an alternate alias name for the table which is open in the current work area. The default alias name is the table name unless you use the AGAIN option. You can also use the letters A through H, or the numbers 1 through 255, to refer to the work areas.

AUTOMEM Initializes a memory variable for each field of the specified table (not including memo, binary, or OLE types). The memory variables are assigned the same names and types as the fields.

EXCLUSIVE | SHARED EXCLUSIVE opens the table so that no other users can open the table until you close it; SHARED allows other users access while the table is opened. The default setting depends on whether you are accessing tables in a shared environment or the LOCALSHARE setting in a single-user environment.

NOSAVE Used to open a .DBF file as a temporary table. When you close a file opened with NOSAVE, it is erased along with its associated memo file. If you inadvertently open a table with the NOSAVE option, use COPY TO to save the data.

NOUPDATE Prevents users from altering, deleting, or recalling any records in the table.

Description

The USE command opens an existing table, production (.MDX) index file, and associated .DBT file if the table contains binary, memo, or OLE fields). Optionally, USE also opens any associated .NDX or non-production .MDX index files that you specify. You need to open a table before you can access any data stored in the table.

USE with no other argument closes the open table, indexes, and format file in the current work area. USE IN <alias>, with no file name argument, closes the table and related files in the specified work area. CLOSE TABLES closes tables in all work areas.

You can open a table in any available work area. When opening a table, you can name the work area by including the ALIAS option in the USE command line. ALIAS names follow the same rules as file names. Aliases are used when referring to a table from another work area.

USE...INDEX specifies index files that are opened and maintained for a particular table. The ORDER option specifies the master index from the list of indexes opened with the INDEX option and the production .MDX index. USE...INDEX is identical to USE followed by SET INDEX. See the SET INDEX and SET ORDER commands for an explanation of the open index order and specifying a master index.

You can include .NDX as well as .MDX index file names with the INDEX option. If a table has an .NDX and an .MDX index file with the same name, *Visual* dBASE opens indexes listed in the .MDX index file. In that case, to open the single index you would need to specify its full name, including its extension.

Use the NOSAVE option of USE to open a table as a temporary file. *Visual* dBASE automatically erases the table, along with the associated memo and index files listed previously, when you close the table.

USE

The AUTOMEM option, which creates memory variables for all fields of the table (excluding binary, memo, and OLE type fields), initializes the memory variables according to the data type in the field as follows:

Character	All blanks
Float	0
Numeric	0
Logical	.F.
Date	/ /

USE...AUTOMEM lets you create memory variables corresponding to fields in a table for updating data values in an application program. See the APPEND AUTOMEM and STORE AUTOMEM for more information on using automem variables.

Example

The following example demonstrates USE to open Flights and Aircrdb tables ordered by an .MDX tag in the next two available work areas:

```
CLOSE DATABASES
USE Flights ORDER Aircraft EXCLUSIVE IN SELECT()
USE Aircrdb IN SELECT()
SELECT Aircrdb
SET RELATION TO Aircraft INTO Flights
```

The following example opens the Clients table and automatically creates memory variables from each of the field values:

```
SELECT 1
USE Clients ORDER Company AUTOMEM EXCLUSIVE
DISPLAY MEMORY TO PRINT
* Displays memory variables created with AUTOMEM from Clients table.
CLOSE DATABASES
```

The following example opens a Paradox table:

```
USE Customer TYPE PARADOX
BROWSE
CLOSE DATABASES
```

See Also

ALIAS(), APPEND AUTOMEM, CLEAR AUTOMEM, CLOSE..., SELECT, SELECT(), SET INDEX, SET ORDER, STORE AUTOMEM

USER()

Returns the login name of the user currently logged in to a protected system.

Syntax

USER()

Description

The USER() 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

ACCESS(), PROTECT

VAL()

Returns a specified character string as a numeric or float value.

Syntax

VAL(<expC>)

<expC> The character expression to return as a numeric or float value, or the character expression *beginning* with the number to return as a numeric or float value.

Description

Use VAL() to convert character expressions to numbers of numeric or float type. Once you convert character data to numeric or float data, you can perform arithmetic operations with it.

If the character string you specify contains both letters and numbers, VAL() returns the value of the entire number to the left of the first nonnumeric character. If the string contains a nonnumeric character other than a blank space in the first position, VAL() returns 0. For example, VAL("ABC123ABC456") returns 0, VAL("123ABC456ABC") returns 123, and VAL(" 123") also returns 123.

Example

The following examples use VAL() to return the numeric value of a character string, variable or field.

```
SET TALK OFF
? VAL("123AB34")           && Returns 123
? VAL("-32")               && Returns -32
Strng1 = "BAY672"
? VAL(Strng1)              && Returns 0
Strng2 = "345 Alcott Lane"
? VAL(Strng2)              && Returns 345
```

VALIDDRIVE()

```
**FldDemo.Prg**
Strng2 = "345 Alcott Lane"
USE Clients ORDER Company
SEEK "Brown Designs"
IF FOUND()
    REPLACE ADDRESS WITH Strng2
ENDIF
? VAL(Address)                && Returns 345
SET TALK ON
CLOSE ALL
** End FldDemo.PRG**
```

The following example uses VAL() to convert character data in the Zip_P_Code field to numeric type to filter only those records in the western region of Clients table.

```
SET TALK OFF
USE Clients EXCLUSIVE
INDEX ON Zip_P_Code TAG ZIP
SET FILTER TO VAL(Zip_P_Code)>80000
GO TOP
? CENTER("Phone List")
? CENTER("Western Region")
?
SCAN
    ? Company, Contact, Areacode, Phone, Extension
ENDSCAN
CLOSE ALL
SET TALK ON
RETURN
```

See Also

SET DECIMALS, SET POINT, SET SEPARATOR, STR()

VALIDDRIVE()

Disk and file utilities

Returns .T. if the specified drive exists and can be read. Returns .F. if the specified drive does not exist or cannot be read.

Syntax

VALIDDRIVE(<drive expC>)

<drive expC> Drive letter of the drive to be tested, optionally followed by a colon. If you specify more than one letter in <drive expC>, dBASE evaluates only the drive represented by the first letter.

Description

Use VALIDDRIVE() to determine if a specified drive exists and is ready before using CD, SET DEFAULT, SET DIRECTORY or SET PATH. VALIDDRIVE() is also useful if your program copies files to or from a drive, or includes drive letters in any file names.

VALIDDRIVE() can verify any drive specified, including drives created by partitioning a hard disk and network drives.

Example

The following examples use VALIDDRIVE():

```
? VALIDDRIVE("C:")           && .T.
? VALIDDRIVE("A:")           && .T. if A drive has floppy ready
? VALIDDRIVE("Z:")           && .T. if Z drive exists
```

In the following example, VALIDDRIVE() checks a floppy drive, the B drive. If VALIDDRIVE() returns false then the user is asked to insert a floppy disk. This code will loop three times.

```
K=1                               && set up a counter
DO WHILE .NOT. VALIDDRIVE("B:") .AND. K<=3
    WAIT "Put a disk in drive B and press any key"
    K=K+1                           && increment the counter
ENDDO
```

Portability

Not supported in dBASE IV or dBASE III PLUS.

See Also

CD, SET DEFAULT, SET DIRECTORY, SET PATH

VARREAD()

Input/Output

Returns the name of the current memory variable or field when you issue READ after @...GET. This command is supported primarily for compatibility with dBASE IV. In *Visual* dBASE, use the Name property to identify an object in a form.

For complete syntax information on VARREAD(), see online Help. For more information about working with forms, see the Forms chapters in the *User's Guide*.

VERSION()

Environment

Returns a character string that is the name and version number of the currently running version of dBASE.

Syntax

VERSION()

Description

Use VERSION() in programs that use features specific to different versions of dBASE. For example, you can use VERSION() to learn what version of dBASE is in use and then execute code specific to that version.

Example

The following example tests if the user is running *Visual* dBASE:

U-V

```

IF "WINDOW"$UPPER(VERSION())
* Does "WINDOW" occur in the version name?
? "Running Windows"                                && Yes
ELSE
* User is not running a Windows version of dBASE
WAIT "You must have Visual dBASE"+ "to run this program"
ENDIF

```

See Also

OS()

WAIT

Input/Output

Pauses the current program, optionally accepts a single character as data input, and continues execution when any key is pressed. This command is supported primarily for compatibility with dBASE IV. In *Visual* dBASE, WAIT is useful only if you are sending output to the results pane of the Command window and want to pause at particular points.

Syntax

```

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

```

<prompt expC> A character expression that prompts the user for input.

TO <memvar> Assigns a single character to the memory variable you specify for <memvar> as a character-type variable. If <memvar> doesn't exist, dBASE creates it. If <memvar> does exist, WAIT overwrites it.

Default

If you don't specify <prompt expC>, dBASE displays "Press any key to continue" when you issue WAIT.

Description

Use WAIT to halt program execution temporarily (for example, to display data without enabling editing). Pressing any key exits WAIT and resumes program execution.

Note If SET ESCAPE is ON, pressing *Esc* at the WAIT prompt causes dBASE to interrupt program execution. If SET ESCAPE is OFF, pressing *Esc* in response to WAIT causes program execution to resume the same as any other key.

Although you use WAIT primarily to pause a program until the user presses a key, you can also use it to store the pressed key to a character memory variable. If the user presses *Enter* without typing any characters, WAIT assigns an empty string ("") to <memvar>.

Example

Wait can be used without a prompt or an input variable:

```

WAIT

```

The default prompt then appears on the next row, in column 0:

```
* Press any key to continue ...
```

The following example shows WAIT with a prompt:

```
WAIT "Press any key to move to the next screen"
```

This examples shows WAIT with a prompt and input variable:

```
WAIT "Do you wish to print the report (Y/N)? " to Answer
```

See Also

SET ESCAPE

WINDOW()

dBASE IV windows

Returns the name of the active dBASE IV-style window. This command is supported primarily for compatibility with dBASE IV. In *Visual dBASE*, use INSPECT() to return information associated with forms.

For complete syntax information on WINDOW(), see online Help. For information about defining forms, see the Forms chapters in the *User's Guide*.

WORKAREA()

Table basics

Returns a number representing the currently selected work area.

Syntax

WORKAREA()

Description

The WORKAREA() function returns the number of the currently selected work area. Use WORKAREA() in a program to save the current work area number and then later restore that work area using the SELECT command.

Example

See DBF() and SELECT() for an examples of using WORKAREA().

Portability

Not supported in dBASE IV or dBASE III PLUS.

See Also

DBF(), SELECT, SELECT()

YEAR()

Returns the year of a specified date expression as a 4-digit number.

Syntax

YEAR(<expD>)

<expD> The date expression, in the current date format, whose corresponding year number to return.

Description

YEAR() returns a date's 4-digit year number. The SET CENTURY setting has no effect on YEAR().

Enter <expD> in the current date format as determined by SET DATE, DBASEWIN.INI, or the International option of the Windows Control Panel (in that order). That is, SET DATE settings override those in DBASEWIN.INI, and settings in DBASEWIN.INI override those in the Windows Control Panel. Be sure <expD> matches the date format in use when your program runs.

If you pass an invalid date to YEAR(), dBASE converts the date to a valid one and returns the 4-digit year of that date. If you pass an empty or non-date expression delimited with braces ({ }) to YEAR(), it returns 0. If you pass a non-date expression or an expression that isn't delimited with braces to YEAR(), it returns an error.

You can use STR(YEAR()) to index on a date field in combination with a character field.

Example

See the example of CMONTH() for the use of YEAR().

See Also

DATE(), DAY(), DOW(), MONTH(), SET CENTURY, STR()

ZAP

Removes all records from the current table.

Syntax

ZAP

Description

ZAP is the fastest way to delete all records from a table. DELETE ALL, followed by PACK, also deletes all records from a table. Using ZAP requires a table be opened exclusively.

When SET SAFETY is ON and you issue ZAP, *Visual* dBASE displays a warning message asking you to confirm the operation before removing records.

Example

The following example uses ZAP to *permanently* remove all records from the current table:

```
USE Clients
SET TALK OFF
SET SAFETY ON
COPY TO TEMP
USE TEMP EXCLUSIVE
zap_yn = "N"
ACCEPT "Do you want to remove all records? (Y/N)" TO zap_yn
READ
IF UPPER(zap_yn) = "Y"
    ZAP
    ? "All records have been removed from " + ALIAS()
ENDIF
CLOSE ALL
SET TALK ON
```

See Also

DELETE, PACK, SET SAFETY



System memory variables

System memory variables

_alignment

Printing

Left-aligns, right-aligns, or centers ? and ?? command output within margins specified by `_lmargin` and `_rmargin` when `_wrap` is true.

Syntax

`_alignment = <expC>`

<expC> The character expression "LEFT", "CENTER", or "RIGHT". You can enter **<expC>** in any combination of uppercase and lowercase letters.

Default

The default for `_alignment` is "LEFT".

Description

Use `_alignment` to left-align, right-align, or center output from the ? and ?? commands between the margins you set with `_lmargin` and `_rmargin`. The `_alignment` setting is effective only when `_wrap` is true (.T.).

To control the alignment of text within a field, use the "B," "I," and "J" format options with the PICTURE or FUNCTION options of an @...SAY statement. For information about format options, see the Picture and Function sections in Chapter 8.

Example

The following example sets wrap on and then prints in the three different alignments: left, center, and right:

```
savewrap=_wrap      && save last wrap setting
_wrap=.t.           && must be .t. for alignment
savealign=_alignment && save last alignment setting
_alignment="LEFT"
? "Hello LEFT"
```

```
_alignment="RIGHT"  
? "Hello RIGHT"  
_alignment="CENTER"  
? "Hello CENTER"  
_alignment=savealign && reset  
_wrap=savewrap      && reset
```

Portability

Not supported in dBASE III PLUS.

See Also

?, ??, @...SAY...GET, `_lmargin`, `_rmargin`, `_wrap`, SET MARGIN

`_app`

Objects

Contains an object reference to the application object, which is the active dBASE session.

Syntax

`_app.<property name> = <new value>`

<property name> The application object property that you change.

<new value> The value you give to the application object property.

Description

Use `_app` to access and control the application object.

The application object is the current dBASE session. It has four properties, `DdeServiceName`, `ClassName`, `Insert`, and `OnInitiate`. The application object also contains an object called `FrameWin`, which is the dBASE application window.

DdeServiceName

Use the `DdeServiceName` property to let a client application establish a DDE link to a particular dBASE session when multiple dBASE sessions exist.

The value you specify with `DdeServiceName` is the name of a dBASE application object. Any active dBASE session is an application object; when you start a dBASE session, you are actually creating an instance of a dBASE application object.

For example, when you start two dBASE sessions, you create two dBASE application objects. Setting the `DdeServiceName` property of one application object to "NEWSERV" (or some other unique name) lets an external application like Quattro Pro distinguish between the two sessions. You use the `_app` system memory variable to reference and set the `DdeServiceName` property of one of the sessions:

```
_app.DdeServiceName = "NEWSERV"
```

To create a DDE link to the NEWSERV session, Quattro Pro could execute the following {INITIATE} request:

```
{INITIATE "NEWSERV", "MyTopic", CHANNEL}
```

To create a DDE link to the other session (whose DdeServiceName property still contains the default value "DBASEWIN"), Quattro Pro could execute the following {INITIATE} request:

```
{INITIATE "DBASEWIN", "MyTopic", CHANNEL}
```

ClassName

ClassName identifies the application object's class (APPLICATION). Classname is a read-only property.

Insert

Determines if *Insert* is on or off. When you set Insert to true (.T.), *Insert* is on.

OnInitiate

Executes an initiation-handler subroutine. You write this routine to create DDETopic objects, which handle DDE server events. OnInitiate executes the subroutine whenever a client application requests a DDE link with the dBASE session. For more information, see CLASS DDETopic and Chapter 26 in the *Programmer's Guide*.

The FrameWin object

The dBASE application window. This object contains five properties, Visible, ClassName, hWnd, Text, and WindowState.

Visible Determines whether the dBASE application window is visible or hidden. You set Visible to false (.F.) when you want to display non-MDI forms without displaying the dBASE environment.

ClassName ClassName identifies the application object's class (FRAMEWINDOW).

hWnd Returns the dBASE Frame Window's object handle. External functions written in other languages such as C, Pascal, or ASM use this handle to identify the object. These external functions are usually stored in Dynamic Link Library (DLL) files. For example, you can pass the object handle of the dBASE Frame Window as a parameter to an external function, perhaps allowing the function to open or close the window.

hWnd is a read-only value.

For more information on DLL files and external functions, see EXTERN and Chapter 25 in the *Programmer's Guide*.

Text Specifies a character string to display in the title bar of the dBASE application window. For more information, see the description of the Text property.

WindowState Determines if the dBASE application window is minimized, maximized, or displayed in its original size. For more information, see WindowState.

Example

The following command, placed at the beginning of an application program, would change the text in the upper-center of the dBASE application window to a company name:

```
_app.FrameWin.Text = "National Federal Bank"
```

To remove the dBASE environment from the screen:

```
_app.FrameWin.Visible = .F.
```

To set Insert Off for your application:

```
_app.Insert = .F.
```

Since the `ClassName` property is read-only, access it as follows:

```
IF _app.ClassName = "APPLICATION"      && Returns .T.  
    * perform action  
ENDIF  
    * or  
? _app.ClassName && Returns the string "APPLICATION"  
    * or  
mVar = _app.ClassName      && Initializes variable
```

See Also

CLASS DDETopic, EXTERN, LOAD DLL, RESOURCE()

`_box`

Printing

Controls whether dBASE displays dBASE IV DEFINE BOX boxes in ? command output.

Syntax

```
_box = <expL>
```

<expL> The logical expression true or false in the form .T. or .F., respectively.

Default

The default for `_box` is .F.

Description

Use `_box` to control the display of boxes created with the dBASE IV command DEFINE BOX. If you set `_box` to true (.T.), dBASE displays the boxes in the background of streaming output from the ? command.

You can control the exact moment to display or print 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. When you set `_box` to .T. later, the rest of the box prints.

Example

`_box` is used only in conjunction with DEFINE BOX. See the example in DEFINE BOX.

Portability

Not supported in dBASE III PLUS.

See Also

ACTIVATE WINDOW, DEFINE BOX

Identifies the position of the currently selected object in the tabbing order of the active form.

Syntax

_curobj = <expN>

Description

_curobj holds a number identifying the position of the selected object.

A related form property, `ActiveControl`, returns the name of the object that currently has input focus.

Example

The following example defines a form that contains four objects. The `OnGotFocus` property is used to display the current object number at the Command window Results pane when an object gains focus:

```
USE Contact.DBF
CLEAR
LOCAL f
f=NEW ENTRY()
f.OPEN()
CLASS Entry OF FORM
  this.Top=2
  this.Left=2
  this.Width=36
  this.Height=8
  this.Text= "Contact.DBF"
  this.StatusMessage="_curobj in Command window"
  DEFINE ENTRYFIELD CompCode OF THIS AT 2,2;
    PROPERTY Width 5, Height 1.5;;
    DataLink "COMPCODE", OnGotFocus {;?_curobj}
  DEFINE ENTRYFIELD Contact OF THIS AT 2,12;
    PROPERTY Width 22, Height 1.5;;
    DataLink "CONTACT", OnGotFocus {;?_curobj}
  DEFINE PUSHBUTTON Back OF THIS AT 5,9;
    PROPERTY Text "Back", Height 2;;
    OnClick {;SKIP-1},;
    OnGotFocus {;?_curobj}
  DEFINE PUSHBUTTON Next OF THIS AT 5,19;
    PROPERTY TEXT "Next", Height 2;;
    OnClick {;SKIP},;
    OnGotFocus {;?_curobj}
ENDCLASS
```

Portability

Not supported in dBASE IV or dBASE III PLUS.

See Also

ActiveControl, NEXTOBJ(), SET FOCUS TO

`_dbwinhome`

Disk and file utilities

Contains the path to the dBASE home directory.

Syntax

? `_dbwinhome`

Description

Use `_dbwinhome` to identify the root directory (also called the home directory) in which dBASE files are installed. When you install dBASE, the installation program (by default) installs dBASE files in the DOS directory `\VISUALDB`, and creates subdirectories such as `\BIN` and `\SAMPLES` under `\VISUALDB`. `_dbwinhome` returns the name of the home directory (in this case, `DBASEWIN.`) `_dbwinhome` returns the full path name whether `SET FULLPATH` is ON or OFF.

To identify where the currently running version of `DBASEWIN.EXE` is located, use `HOME()`.

Example

`_DBWINHOME` would only change if dBASE were reinstalled in a different directory:

```
? _dbwinhome && e.g. D:\VISUALDB
* dBASE is installed in the VISUALDB subdirectory
* of the D drive.
```

Portability

Not supported in dBASE IV or dBASE III PLUS.

See Also

CD, HOME(), MKDIR, SET DIRECTORY, SET FULLPATH, SET PATH

`_indent`

Printing

Specifies the number of columns to indent the first line of a paragraph of ? command output when `_wrap` is true.

Syntax

`_indent = <expN>`

<expN> The column number, relative to the left margin, where the first line of a new paragraph begins. You can specify a fractional number for `<expN>` to position output accurately with a proportional font.

Default

The default for `_indent` is 0.

Description

Use `_indent` to specify where the first line of a new paragraph begins relative to the left margin. (Specify the left margin with `_lmargin`.) The `_indent` setting is effective only when `_wrap` is true (.T.).

When you direct ? output to the printer, dBASE maps each character according to the *coordinate plane*, a two-dimensional grid. The coordinate plane is divided into character cells whose widths depend on the value of `_ppitch`. See the table in the description of `_ppitch`, which lists `_ppitch` values. The height of each character cell is determined by the size of the font. For more information about the coordinate plane, see Chapter 16 in the *Programmer's Guide*.

If you change the value of `_indent`, dBASE takes the current value of `_ppitch` into consideration when calculating the cell width of the coordinate plane. This happens regardless of whether you're printing with proportional or monospaced fonts. If you issue ? without the STYLE option and use only integer coordinates, dBASE uses a monospaced font, and all output appears exactly the same as in dBASE IV.

To indent the first line of a paragraph, use a value greater than 0. For example, to begin the line five columns to the right of the left margin, set `_indent` to 5. To create a hanging indent (sometimes called an *outdent*), use a negative value. For example, to begin the first line five columns to the *left* of the left margin, set `_indent` to -5. Using the default value of 0 (no indent or outdent) aligns all lines in a paragraph to the left margin. The sum of `_lmargin` and `_indent` must be greater than 0 and less than `_rmargin`.

Example

The following example sets wrap on and indents the first line of a text that wraps around:

```

_indent=3           && set the indentation
savewrap=_wrap      && save last wrap setting
_wrap=.t.           && must be .t. for alignment
savelmarg=_lmargin  && save last alignment setting
_lmargin=5
savermarg=_rmargin  && save last alignment setting
_rmargin=20
? "New York, Chicago and Boston are "+;
  "cold in wintertime."
* Now the text wraps around between columns 5 and 20
*
*      New York,
*      Chicago and
*      Boston are cold
*      in wintertime.
*
_rmargin=savermarg  && restore the previous margin
_lmargin=savelmarg  && restore the previous margin
_wrap=savewrap      && reset wrap

```

Portability

Not supported in dBASE III PLUS.

See Also

?, ??, _alignment, _lmargin, _ploffset, _rmargin, _wrap, SET MARGIN

_lmargin

Printing

Defines the left margin for ? and ?? command output when _wrap is true.

Syntax

_lmargin = <expN>

<expN> The column number of the left margin. The valid range is 0 to 254, inclusive. You can specify a fractional number for <expN> to position output accurately with a proportional font.

Default

The default for _lmargin is 0.

Description

Use _lmargin to set the left margin for ? and ?? command output. If you're sending output to a printer, _lmargin sets the left margin from the _ploffset (page left offset) column. For example, if _ploffset is 10 and _lmargin is 5, output prints from the 15th column. The _lmargin setting is effective only when _wrap is true (.T.).

When you direct output to the printer, dBASE maps each character according to the *coordinate plane*, a two-dimensional grid. The coordinate plane is divided into character cells whose widths depend on the value of _ppitch. See the table in the description of _ppitch, which lists _ppitch values. The height of each character cell is determined by the size of the font. For more information about the coordinate plane, see Chapter 16 in the *Programmer's Guide*.

If you change the value of _lmargin, dBASE takes the current value of _ppitch into consideration when calculating the cell width of the coordinate plane. This happens regardless of whether you're printing with proportional or monospaced fonts. If you issue ? without the STYLE option and use only integer coordinates, dBASE uses a monospaced font, and all output appears exactly the same as in dBASE IV.

If you use _indent to specify the indentation of the first line of each paragraph, the combined values of _lmargin and _indent must be less than the value of _rmargin.

Example

The following example uses _lmargin. It sets wrap on and then changes the left margin and displays text:

```
_wrap=.t.           && must be .t. for _lmargin
_lmargin=0
? "01234567890"
```



```
_lmargin=5
? "Changing the margin"
* produces:
* 01234567890
*      Changing the margin
```

Portability

Not supported in dBASE III PLUS.

See Also

?, ??, `_alignment`, `_indent`, `_ploffset`, `_rmargin`, `_wrap`, SET MARGIN, Style

`_padvance`

Printing

Determines whether the printer advances the paper of a print job with a formfeed or with linefeeds.

Syntax

`_padvance = <expC>`

<expC> The character expression "FORMFEED" or "LINEFEEDS".

Default

The default for `_padvance` is "FORMFEED".

Description

Use `_padvance` to specify whether dBASE advances the paper to the top of the next sheet one sheet at a time using a formfeed character, or one line at a time using linefeed characters. If you use the default "FORMFEED" setting, the paper advances according to the printer's default form length setting.

Tractor-feed printers (such as dot matrix printers) generally use a "LINEFEEDS" setting, while form feed printers (such as laser printers) generally use a "FORMFEED" setting.

Note Sending `CHR(12)` to the printer always issues a formfeed, even if you set `_padvance` to "LINEFEEDS".

Use the "LINEFEEDS" setting if you change the length of the paper or want to print a different number of lines than the default form length of the printer without adjusting its setting. For example, to print short pages, such as checks that are 20 lines long, set `_plength` to the length of the output (20 in this example) and `_padvance` to "LINEFEEDS."

The number of linefeeds dBASE uses to reach the top of the next page depends on whether you issue an eject during streaming or non-streaming output mode. For more information about streaming and non-streaming output, see Chapter 24 in the *Programmer's Guide*.

An eject occurs during streaming output mode when you issue:

- EJECT PAGE without an ON PAGE handler
- EJECT PAGE with an ON PAGE handler when the current line position is past the ON PAGE line
- PRINTJOB or ENDPRINTJOB and `_peject` causes an eject

In these cases, dBASE calculates the number of linefeeds to send to the print device using the formula `_plength - _plineno`.

An eject occurs in nonstreaming output mode when you issue:

- EJECT
- SET DEVICE TO PRINTER and force a page eject with the @ command

In these cases, dBASE calculates the number of linefeeds to send to the print device using the formula `_plength - MOD(PROW(), _plength)`.

Example

There are two `_padvance` settings:

```
_padvance="FORMFEED"  
_padvance="LINEFEEDS"
```

Portability

Not supported in dBASE III PLUS.

See Also

`_peject`, `_plength`, EJECT, EJECT PAGE, PRINTJOB...ENDPRINTJOB, ON PAGE, SET DEVICE

`_pageno`

Printing

Determines or sets the current page number.

Syntax

`_pageno = <expN>`

<expN> An integer from 1 to 32,767, inclusive.

Description

Use `_pageno` to number pages of *streaming output* from commands such as `?`, `??`, and `LIST`. For more information about streaming and non-streaming output, see Chapter 24 in the *Programmer's Guide*.

With `_pageno`, you can determine the current page number or set the page number to a specific value. Use it to print page numbers in a report or, when combining documents, to assign an incremented number to the first page of the second document.

A page break occurs when the value of `_plineno` (the line number count) becomes greater than the value of `_plength` (the currently defined printed page length in lines).

At each page break of streaming output, dBASE automatically increments the value of _pageno.

Example

This example prints 100 lines of output and prints a heading on line 1 of each page:

```
SET TALK OFF
SET PRINTER ON
_pageno=1
FOR i=1 TO 100
  IF _plineno=1  && At first line of page
    ? "Top of Page ",_pageno
  ENDIF
  ? "Line",i
NEXT i
SET PRINTER TO
CLOSE PRINTER
SET TALK ON
```

Portability

Not supported in dBASE III PLUS.

See Also

?, ??, _pbpage, _pepage, _plength, _plineno, LIST, ON PAGE

_pbpage

Printing

Specifies the page number of the first page PRINTJOB prints.

Syntax

_pbpage = <expN>

<expN> The page number at which to begin printing. The valid range is 1 to 32,767, inclusive. Specify a positive integer for <expN>.

Default

The default for _pbpage is 1.

Description

Use _pbpage to begin printing a print job at a specific page number. Pages with numbers less than _pbpage don't print. To stop printing at a specific page number, use _pepage.

If you set _pbpage to a value greater than _pepage, dBASE returns an error.

Example

This example uses `_pbpage` to omit a page of a report. It outputs 100 lines and prints the page and line number on each line as in the example for `_plineno`. Here, the beginning page number is set to 2 so that page 1 does not print:

```
_pageno=1
_pbpage=2      && begin on page 2
SET PRINTER ON
PRINTJOB
FOR i=1 TO 100
  ?? "Page",_pageno," Line",_plineno
  ?              && now force a linefeed
NEXT i
ENDPRINTJOB
SET PRINTER OFF
CLOSE PRINTER      && start printing
```

Portability

Not supported in dBASE III PLUS.

See Also

`_pageno`, `_page`, `PRINTJOB...ENDPRINTJOB`

`_pcolno`

Printing

Identifies or sets the current column number of streaming output.

Syntax

`_pcolno = <expN>`

<expN> The column number at which to begin printing. The valid range is 0 to 255, inclusive. You can specify a fractional number for `<expN>` to position output accurately with a proportional font.

Default

The default for `_pcolno` is 0.

Description

Use `_pcolno` to position printing streaming output from commands such as `?`, `??`, and `LIST`. For more information about streaming output, see Chapter 24 in the *Programmer's Guide*.

When you direct output to the printer, dBASE maps each character according to the *coordinate plane*, a two-dimensional grid. The coordinate plane is divided into character cells whose widths depend on the value of `_ppitch`. See the table in the description of `_ppitch`, which lists `_ppitch` values. The height of each character cell is determined by the size of the font. For more information about the coordinate plane, see Chapter 16 in the *Programmer's Guide*.

If you change the value of `_pcolno`, dBASE takes the current value of `_ppitch` into consideration when calculating the cell width of the coordinate plane. This happens regardless of whether you're printing with proportional or monospaced fonts. If you issue ? without the `STYLE` option and use only integer coordinates, dBASE uses a monospaced font, and all output appears exactly the same as in dBASE IV.

The `PCOL()` function also returns the current printhead position of the printer, but if `SET PRINTER` is `OFF`, the `PCOL()` value doesn't change. `_pcolno`, on the other hand, returns or assigns the current position in the streaming output regardless of the `SET PRINTER` setting.

Example

This example displays the numbers 1 through 5. It uses `_pcolno` to position the numbers so that each number begins at its own position:

```
SET TALK OFF
FOR i=1 to 5
    _pcolno=i      && set the column position
    string=ltrim(str(i))
    * convert i to a single character
    ?? string
    ?
NEXT I
SET TALK ON
* the output looks like this
* 1
* 2
* 3
* 4
* 5
```

Portability

Not supported in dBASE III PLUS.

See Also

?, `_plineno`, `_rmargin`, `PCOL()`, `SET DEVICE`, `SET PRINTER`

_pcopies

Printing

Specifies the number of copies to print for a `PRINTJOB`.

Syntax

`_pcopies = <expN>`

<expN> The number of copies to print. The valid range is 1 to 32,767, inclusive. Specify a positive integer for `<expN>`.

Default

The default for `_pcopies` is 1.

Description

Use _pcopies to print a specific number of copies of a print job. You can assign a value to _pcopies in the Command window or in a program. The value of _pcopies has an effect only when you send a print job to the printer by issuing PRINTJOB. In a program, assign a value to _pcopies before issuing PRINTJOB.

Example

This example uses _pcopies to print the print job three times:

```
_pcopies=3          && Three copies
SET PRINTER ON
PRINTJOB
? "Very Small Report"
ENDPRINTJOB
SET PRINTER OFF
CLOSE PRINTER      && start the printer
```

Portability

Not supported in dBASE III PLUS.

See Also

PRINTJOB...ENDPRINTJOB

_pdriver

Printing

Identifies the current printer driver or activates a new driver.

Syntax

_pdriver = <expC>

<expC> The name of the printer driver to activate.

Default

The default for _pdriver is the printer driver you specify with the Windows Control Panel. If you haven't specified a printer driver with the Control Panel, the value of _pdriver is an empty string ("").

Description

Use _pdriver to identify the current printer driver or to activate an installed driver. (To install a new printer driver, use the Windows Control Panel.)

The _pdriver value contains two elements separated by a comma: the base file name of the Windows driver file and the name of the printer as it appears in WIN.INI. The current driver might not identify a printer name, in which case, _pdriver contains only the driver file name. For example, if the current printer driver is for the HP LaserJet IIISi PostScript printer, _pdriver may contain the value "pscript,HP LaserJet IIISi PostScript". To activate this driver, issue the command _pdriver = "pscript,HP LaserJet IIISi PostScript".

To activate a driver from within dBASE, it may be easier to use CHOOSEPRINTER() than to assign a value to _pdriver. To activate a driver in Windows, use the Printers program of the Windows Control Panel. CHOOSEPRINTER() opens the Print Setup dialog box, in which you can also select options such as paper size, source, and orientation (portrait or landscape). In the Windows Control Panel, you can choose Setup to select these options.

Example

Use _pdriver to determine the current print driver:

```
? _pdriver
* With an Epson FX 80, the response is:
*   EPSON9,Epson FX-80
* With an HP Laserjet running postscript, the
* response is:
*   pscript,HP LaserJet IIISi PostScript
```

You can set the print driver with _pdriver:

```
_pdriver="pscript"
```

Portability

Not supported in dBASE III PLUS. Only Windows printer driver names are supported; driver names used with dBASE IV are not supported.

See Also

_pform, _ppitch, _pquality, CHOOSEPRINTER(), SET PRINTER

_pject

Printing

Determines whether dBASE ejects a sheet of paper before and after a PRINTJOB.

Syntax

```
_pject = <expC>
```

<expC> The character expression "before", "after", "both", or "none".

Default

The default for _pject is "before", which tells the printer to eject a sheet of paper before starting the print job.

Description

Use _pject to specify if and when the printer should eject a sheet of paper. Assign a new value to _pject (and to any other system memory variable) before issuing PRINTJOB in a program to make the new value affect the print job.

The following table describes _peject options.

<expC>	Result
"before"	Eject sheet before printing the first page
"after"	Eject sheet after printing the last page
"both"	Eject sheet before and after the print job
"none"	Don't eject sheet before or after the print job

Note The _peject system memory variable is distinct from the EJECT command, which tells the printer to advance the paper to the top of the next page.

Example

This example shows the four possible _peject setting. The last setting is operational in the PRINTJOB:

```
_peject="before"
_peject="after"
_peject="both"
_peject="none"
* _peject must be set before PRINTJOB
PRINTJOB
? "Hello World"
ENDPRINTJOB
```

Portability

Not supported in dBASE III PLUS.

See Also

_padvance, EJECT, EJECT PAGE, PRINTJOB...ENDPRINTJOB

_pepage

Printing

Specifies the page number of the last page of a print job.

Syntax

_pepage = <expN>

<expN> The page number of the last page to print. The valid range is 1 to 32,767, inclusive. You must specify a positive integer for <expN>.

Default

The default for _pepage is 32,767.

Description

Use _pepage to stop printing a print job at a specific page number. Pages with numbers greater than _pepage don't print. To begin printing at a specific page number, use _pbpage.

If you set _pepage to a value less than _pbpage, dBASE returns an error.

Example

This example selects two pages from a PRINTJOB. The program prints 500 lines of output and prints the page and line number on each line as in the example for _plineno. Here, the ending page number is set to 2 so that only pages 1 and 2 print:

```
_pageno=1
_pbpage=1      && reset the default pbpage
_pepage=2      && end on page 2
SET PRINTER ON
PRINTJOB
FOR i=1 TO 500
  ?? "Page",_pageno," Line",_plineno
  ?           && now force a linefeed
NEXT i
ENDPRINTJOB
SET PRINTER OFF
CLOSE PRINTER      && begin printing
```

The ? command issues a linefeed before processing consequently in this case, the correct line number is obtained by using ??.

Portability

Not supported in dBASE III PLUS.

See Also

_pageno, _pbpage, PRINTJOB...ENDPRINTJOB

_pform

Printing

Identifies the current print form file or activates another one.

Syntax

_pform = <filename>

<filename> The name of a print form file (.PRF).

Default

The default for _pform is an empty string ("").

Description

Use _pform to determine the name of the current print form file or to activate another one. A print form file (.PRF) is a dBASE binary file that contains print settings for printing a print job. The print form file contains the following system memory variables:

Variable	Action
_padvance	Determines whether the printer advances the paper with a formfeed or linefeeds.
_pageno	Determines or sets the current page number.
_pbpage	Specifies the page number at which PRINTJOB begins printing.
_pcopies	Specifies the number of copies to print in a printjob.
_pdriver	Activates a specified printer driver. (If the print form file is from dBASE IV, <i>Visual</i> dBASE ignores this value.)
_peject	Controls page ejects before and after PRINTJOB.
_pepage	Specifies the number of the last page that PRINTJOB prints.
_plength	Specifies the number of lines per page for streaming output.
_ploffset	Determines the width of the left border on a printed page.
_ppitch	Sets the printer pitch, the number of characters per inch that the printer prints.
_pquality	Specifies if the printer prints in letter-quality or draft mode.
_pspacing	Sets the line spacing for streaming output.

When you specify a print form file by assigning its name to _pform, the values stored in the file are assigned to their respective variables. Jobs you send to the printer then behave in accordance with these variables.

Example

This example assumes there are two reports, Report1 and Report2. It uses Report1's print form file, Report1.PRF to print Report2:

```
_pform= "Report1"  
REPORT FORM Report2
```

Portability

Not supported in dBASE III PLUS.

See Also

PRINTJOB...ENDPRINTJOB

Specifies the number of lines per page for streaming output.

Syntax

_p length = <expN>

<expN> The number of lines per page. The valid range is 1 to 32,767, inclusive. You can specify a fractional number for <expN> to position output accurately with a proportional font.

Default

The default page length is determined by the default page size of the current printer driver and the current page orientation (portrait or landscape).

Description

Use _p length to specify a page length that is different from the default of the current printer driver. For example, to print short pages, such as checks that are 20 lines long, set _p length to the length of the output (20 in this example) and _p advance to "LINEFEEDS" to advance to the top of the next page.

When you change printer drivers or page orientation, dBASE changes the value of _p length automatically. You can change printer drivers in dBASE by issuing CHOOSEPRINTER() or by assigning a value to _p driver. In Windows, you can change printer drivers with the Printers program of the Control Panel (however, it won't take effect until you quit dBASE and start a new dBASE session). You can change page orientation in any of these ways or, in dBASE, by changing the value of _p orientation.

Example

This example sets the form length to 10 lines and prints 25 lines of output. Each line simply prints line number and count (1 through 25). A three-line heading prints "Top of Page" on each line 1:

```
_p length=10
_pageno=1
_plineno=0
SET PRINTER ON
FOR i=1 TO 25
  IF _plino=0 && At first line of page
    ?
    ? "Top of Page ",_pageno
    ?
  ENDIF
  ? "Line",_plino,"i=",i
NEXT i
SET PRINTER OFF
CLOSE PRINTER
* The first two pages are:
*
* Top of Page          1
*
```

`_plineno`

```
* Line      3.00 i=      1
* Line      4.00 i=      2
* Line      5.00 i=      3
* Line      6.00 i=      4
* Line      7.00 i=      5
* Line      8.00 i=      6
* Line      9.00 i=      7
*
* Top of Page      2
*
* Line      3.00 i=      8
* Line      4.00 i=      9
* Line      5.00 i=     10
* Line      6.00 i=     11
* Line      7.00 i=     12
* Line      8.00 i=     13
* Line      9.00 i=     14
```

Portability

Not supported in dBASE III PLUS. In dBASE IV, the default page length is 66 lines.

See Also

`_padvance`, `_pdriver`, `_porientation`, `CHOOSEPRINTER()`, `EJECT`, `EJECT PAGE`

`_plineno`

Printing

Identifies or sets the current line number of streaming output.

Syntax

`_plineno = <expN>`

<expN> The line number at which to begin printing. The valid range is 0 to `_plength - 1`. You can specify a fractional number for `<expN>` to position output accurately with a proportional font.

Default

The default for `_plineno` is 0.

Description

Use `_plineno` to position printing streaming output from commands such as `?`, `??`, and `LIST`. For more information about streaming output, see Chapter 24 in the *Programmer's Guide*.

The `PROW()` function also returns the current printhead position of the printer, but if `SET PRINTER` is `OFF`, the `PROW()` value doesn't change. `_plineno`, on the other hand, returns or assigns the current position in the streaming output regardless of the `SET PRINTER` setting.

Example

This example prints 32 lines of output on four pages. The page length is set at 10 lines per page, and the report prints the page and line number on each line using _pageno and _plinenno:

```
_pbpage=1      && reset the default pbpage
_pepage=32767  && reset the default pepage
_plength=10    && set the page length to 10 lines
_pageno=1
SET PRINTER ON
PRINTJOB       && PRINTJOB resets _plinenno to 0
FOR i=1 TO 32
  IF _plinenno=0 && At first line of page
    *?
    ? "Top of Page ",_pageno
    ?
  ENDIF
  ?? "Page",_pageno," Line",_plinenno,i
  ?                                     && now force a linefeed
NEXT i
ENDPRINTJOB
SET PRINTER OFF
CLOSE PRINTER
```

* The first two pages of output appear as follows:

```
*
* Top of Page          1
* Page      1 Line      2.00      1
* Page      1 Line      3.00      2
* Page      1 Line      4.00      3
* Page      1 Line      5.00      4
* Page      1 Line      6.00      5
* Page      1 Line      7.00      6
* Page      1 Line      8.00      7
* Page      1 Line      9.00      8
*
* Top of Page          2
* Page      2 Line      2.00      9
* Page      2 Line      3.00     10
* Page      2 Line      4.00     11
* Page      2 Line      5.00     12
* Page      2 Line      6.00     13
* Page      2 Line      7.00     14
* Page      2 Line      8.00     15
* Page      2 Line      9.00     16
```

Portability

Not supported in dBASE III PLUS.

See Also

_pcolno, _plength, _ppitch, EJECT PAGE, ON PAGE, PCOL(), PROW()

Displays or sets the width of the left border of a printed page.

Syntax

_ploffset = <expN>

<expN> The column number at which to set the left margin. The valid range is 0 to 254, inclusive. You can specify a fractional number for <expN> to position output accurately with a proportional font.

Default

The default for _ploffset is 0. To change the default, set the MARGIN parameter in DBASEWIN.INI.

Description

Use _ploffset (page left offset) to specify the distance from the left edge of the paper to the left margin of the print area. Use _lmargin to set the left margin from the _ploffset column. For example, if _ploffset is 10 and _lmargin is 5, output prints from the 15th column.

The _ploffset system memory variable is equivalent to the SET MARGIN value. Changing the value of one changes the other. For more information, see SET MARGIN.

Example

See the example in SET MARGIN. _ploffset is identical to SET MARGIN.

Portability

Not supported in dBASE III PLUS, but SET MARGIN is.

See Also

_indent, _lmargin, SET MARGIN

Determines whether the printer prints in portrait or landscape mode.

Syntax

_porientation = <expC>

<expC> The character expression "PORTRAIT" or "LANDSCAPE".

Default

The default for _porientation is the orientation you specify with the Printers program of the Windows Control Panel or, in dBASE, with the CHOOSEPRINTER() function. By default, this orientation is portrait.

Description

Use `_porientation` to specify whether you want to print in portrait or landscape mode. When you print in portrait mode, each page is read vertically; a standard American letter-size piece of paper is 8.5 inches wide by 11 inches long. When you print in landscape mode, each page is read horizontally; a standard American letter-size piece of paper is 11 inches wide by 8.5 inches long.

Most printer drivers support landscape printing; however, if you specify landscape while using a printer driver that doesn't support it, the printer continues to print in portrait mode, possibly truncating text.

Changing page orientation automatically resets `_plength`.

Example

`_porientation` has two settings:

```
_porientation="PORTRAIT"  
_porientation="LANDSCAPE"
```

It takes effect only on a page boundary.

Portability

Not supported in dBASE III PLUS or dBASE IV.

See Also

`_pdriver`, `_plength`, `_ppitch`, SET PRINTER

`_ppitch`

Printing

Sets the printer pitch, the number of characters per inch that the printer prints.

Syntax

`_ppitch = <expC>`

<expC> The character expression "pica", "elite", "condensed", or "default".

Default

The default for `_ppitch` is "default", the pitch defined by your printer's settings or by setup codes or commands you sent to the printer before you started dBASE. "Default" means that dBASE hasn't sent any pitch control codes to the printer.

Description

Use `_ppitch` to set the pitch (characters per inch) on the printer. The `_ppitch` setting sends a control code appropriate to the current printer driver. Use the Windows Control Panel or CHOOSEPRINTER() to select the printer driver.

When you direct output to the printer, dBASE maps each character according to the *coordinate plane*, a two-dimensional grid. The coordinate plane is divided into character cells whose sizes depend on the value of `_ppitch`. The height of each character cell is

determined by the size of the font. For more information about the coordinate plane, see Chapter 16 in the *Programmer's Guide*.

The following table lists _ppitch values.

_ppitch value	Character cell width
"pica"	1/10" (10 characters/inch)
"elite"	1/12" (12 characters/inch)
"condensed"	1/17" (17 characters/inch)

If you change the value in other system memory variables such as _lmargin, _rmargin, and _ploffset, dBASE takes the current value of _ppitch into consideration when calculating the cell width of the coordinate plane. This happens regardless of whether you're printing with proportional or monospaced fonts.

Example

This example shows the three settings of _ppitch:

```
s_ppitch=_ppitch  && save current pitch
SET PRINTER ON
_ppitch="pica"
? "John Brown's body: 10 characters per inch"
_ppitch="elite"
? "John Brown's body: 12 characters per inch"
_ppitch="condensed"
? "John Brown's body: 17 characters per inch"
_ppitch=s_ppitch      && restore original setting
CLOSE PRINTER
```

_ppitch is not valid for all printers.

Portability

Not supported in dBASE III PLUS.

See Also

_pdriver, _pquality, CHOOSEPRINTER()

_pquality

Printing

Specifies whether the printer prints in letter-quality or draft mode. Used primarily with dot-matrix printers; the _pquality value usually has no effect on printers that don't support draft mode, such as laser and Postscript printers.

Syntax

_pquality = <expl>

<expl> The logical expression .T. for letter quality and .F. for draft quality.

Default

The default for _pquality is false (.F.) for draft mode.

Description

Use _pquality to determine whether the printer prints in letter-quality or draft mode. Letter-quality mode produces printed copy of higher quality (finer resolution) than draft; however, draft mode usually prints more quickly than letter-quality, depending on the printer.

Example

This example shows the two settings for print quality. Print quality cannot be changed in mid page and might or might not be available on your printer:

```
CLOSE PRINTER
set printer on
_pquality= .f.      && draft quality
? "John Brown's body"
CLOSE PRINTER
_pquality= .t.      && letter quality
? "John Brown's body"
CLOSE PRINTER
```

Portability

Not supported in dBASE III PLUS.

See Also

_pdriver, _ppitch

_pspacing

Printing

Sets the line spacing for streaming output.

Syntax

_pspacing = <expN>

<expN> The amount of line spacing. The valid range is 1 to 3, inclusive:

- A value of 1 represents single spacing.
- A value of 2 represents double spacing. There is one blank line between printed lines.
- A value of 3 represents triple spacing. There are two blank lines between printed lines.

Paragraph spacing is in multiples of the height of the line just printed, which depends on the tallest font used in printing the line. You can specify a fractional number for <expN> to space text by partial line heights.

Default

The default for _pspacing is 1, which sets line spacing to single-line.

Description

Use `_pspacing` to set the line spacing of streaming output from commands such as `?`, `??`, and `LIST`. For more information about streaming and non-streaming output, see Chapter 24 in the *Programmer's Guide*. To insert a single blank line into output, use the `? command`.

The `_pspacing` value also affects the height of boxes displayed with the dBASE IV `DEFINE BOX` command. For example, if you define a box with a height of 10 and assign 2 to `_pspacing`, dBASE prints the box with a height of 20 lines.

Example

This example uses `_pspacing` to set the spacing to 1 then 2 lines between lines of text and finally back to 1 line:

```
_pspacing=1
? "Jack 1"
? "Jill 1"
_pspacing=2
? "Jack 2"
? "Jill 2"
_pspacing=1
? "Jack 1"
? "Jill 1"
* produces:
*      Jack 1
*      Jill 1
*
*      Jack 2
*
*      Jill 2
*      Jack 1
*      Jill 1
```

Notice that `_pspacing` takes place immediately so that the double spacing occurs before Jack 2 and before Jill 2.

Portability

Not supported in dBASE III PLUS.

See Also

`?`, `??`, `_ppitch`, `DISPLAY`, `LIST`

_rmargin

Defines the right margin for ? and ?? command output when _wrap is true.

Syntax

_rmargin = <expN>

<expN> The column number of the right margin. The valid range is 0 to 255, inclusive. You can specify a fractional number for <expN> to position output accurately with a proportional font.

Default

The default for _rmargin is 79.

Description

Use _rmargin to set the right margin for output from the ? and ?? commands. The value of _rmargin must be greater than the value of _lmargin or _lmargin + _indent. For example, if _lmargin and _indent are both set to 5, _rmargin must be greater than 10 to display at least one column of output.

When you direct output to the printer, dBASE maps each character according to the *coordinate plane*, a two-dimensional grid. The coordinate plane is divided into character cells whose widths depend on the value of _ppitch. See the table in the description of _ppitch, which lists _ppitch values. The height of each character cell is determined by the size of the font. For more information about the coordinate plane, see Chapter 16 in the *Programmer's Guide*.

If you change the value of _rmargin, dBASE takes the current value of _ppitch into consideration when calculating the cell width of the coordinate plane. This happens regardless of whether you're printing with proportional or monospaced fonts. If you issue ? without the STYLE option and use only integer coordinates, dBASE uses a monospaced font, and all output appears exactly the same as in dBASE IV.

Example

The following example sets wrap on and then changes the left margin and displays text:

```
savewrap=_wrap           && save last wrap setting
_wrap=.t.                && must be .t. for alignment
savelmargin=_lmargin      && save last alignment setting
_lmargin=5
savermargin=_rmargin      && save last alignment setting
_rmargin=20
? "New York, Chicago and Boston are cold in wintertime."
* Now the text wraps around between columns 5 and 20
_rmargin=savermargin      && restore the previous margin
_lmargin=savelmargin      && restore the previous margin
_wrap=savewrap            && reset wrap
```

Portability

Not supported in dBASE III PLUS.

See Also

?, ??, _alignment, _indent, _lmargin, _ploffset, _wrap, SET MARGIN

tabs

Printing

Sets one or more tab stops for output from the ? and ?? commands.

Syntax

_tabs = <expC>

<expC> The list of column numbers for tab stops. If you set more than one tab stop, the numbers must be in ascending order and separated by commas. Enclose the entire list in quotation marks. You can specify fractional numbers for <expC> to position output accurately with a proportional font.

Default

The default for _tabs is an empty string ("").

Description

Use _tabs to define a series of tab stops. If _wrap is true, dBASE ignores tab stops equal to or greater than _rmargin.

When you direct output to the printer, dBASE maps each character according to the *coordinate plane*, a two-dimensional grid. The coordinate plane is divided into character cells whose widths depend on the value of _ppitch. See the table in the description of _ppitch, which lists _ppitch values. The height of each character cell is determined by the size of the font. For more information about the coordinate plane, see Chapter 16 in the *Programmer's Guide*.

If you change the value of _tabs, dBASE takes the current value of _ppitch into consideration when calculating the cell width of the coordinate plane. This happens regardless of whether you're printing with proportional or monospaced fonts. If you issue ? without the STYLE option and use only integer coordinates, dBASE uses a monospaced font, and all output appears exactly the same as in dBASE IV.

If you send a tab character, CHR(9), with ? or ??, dBASE expands it to the amount of space required to reach the next tab stop. If the tab character you send is past the last tab stop, dBASE ignores it, displaying output starting in the current column.

Example

The following program makes two tab stops at columns 5 and 20. The array A is displayed beginning at the first tab stop, and its position in the array (i) is displayed at the second tab stop:

```
_tabs="5,20"    && tab stops at columns 5 and 20
DECLARE A[2]
A[1]="One"
A[2]="Two"
FOR i=1 TO 2
```

```

    ? chr(9),A[i],chr(9),ltrim(str(i))
NEXT i
* chr(9) is equivalent to the tab key
* produces:
*   One           1
*   Two           2

```

Portability

Not supported in dBASE III PLUS. In dBASE IV, the value in `_tabs` also affects the Text Editor. This is not the case in *Visual* dBASE; set tab settings for the Text Editor by setting the Auto Indent value in the Properties Inspector dialog box of the Text Editor window.

See Also

`?, ??, _indent, _lmargin, _rmargin, _wrap, CHR()`, `MODIFY COMMAND`

`_wrap`**Printing**

Determines if streaming output wraps between margins specified by `_lmargin` and `_rmargin`.

Syntax

`_wrap = <expl>`

<expl> The logical expression true or false in the form `.T.` or `.F.`, respectively.

Default

The default for `_wrap` is `false (.F.)`, which disables wordwrapping.

Description

Set `_wrap` to `.T.` to wrap streaming output from commands such as `?, ??`, and `LIST` within the margins you specify with `_lmargin` and `_rmargin`. For more information about streaming output, see Chapter 24 in the *Programmer's Guide*.

When you enable wordwrapping, dBASE wraps text onto the next line, breaking between words or numbers, when the output reaches the right margin. When you disable wordwrap mode, dBASE extends text beyond the right margin, moving to the next line only when a carriage return and linefeed combination (CR/LF) occurs in the text.

The value of `_wrap` must be `.T.` for `_alignment`, `_indent`, `_lmargin`, and `_rmargin` to have an effect.

When `_wrap` is `.T.`, dBASE stores streaming output in a buffer until it finishes displaying or printing the current line. If you generate output with the `?` command, follow it with another `?` command to force the last line of text to print.

Example

This example shows a long string displayed with wrap on and with wrap off:

`_wrap`

```
_lmargin=5
_rmargi=15
string="Now is the time for all men and women to come to."
_wrap=.f.
? string
_wrap=.t.
? string
* wrap false displays as:
* Now is the time for all men and women to come to.
* wrap true displays as:
*   Now is the
*   time for
*   all men and
*   women to
*   come to.
```

Portability

Not supported in dBASE III PLUS. In dBASE IV, the value in `_wrap` also affects the Text Editor. This is not the case in *Visual* dBASE; set wrap on or off for the Text Editor by selecting or deselecting the Word Wrap check box in the Properties Inspector dialog box of the Text Editor window.

See Also

?, ??, `_alignment`, `_indent`, `_lmargin`, `_ploffset`, `_rmargin`, `PRINTJOB...ENDPRINTJOB`



Preprocessor directives

Preprocessor directives

#define

Preprocessor

Defines an identifier (name) for use in controlling program compilation, defining constants, or creating inline functions.

Syntax

```
#define <identifier> [<replacement text>]
```

```
#define <identifier>(<parameter list>) <replacement text with parameters>
```

<identifier> A name. It identifies the text to replace if *<replacement text>* is supplied. The name must start with an alphabetic character and can contain any combination of alphabetic or numeric characters, uppercase or lowercase letters. The identifier is not case-sensitive.

<parameter list> Parameter names that correspond to arguments passed to an inline function (pseudo function) that you create with `#define <identifier> (<parameter list>) <replacement text>`. If you specify multiple parameters, separate each with a comma.

<replacement text> The text that replaces all occurrences of *<identifier>*. If you specify *<replacement text>*, the preprocessor scans each source code line for identifiers and replaces each one it encounters with the specified replacement text. *<replacement text>* can be any text that is part of a dBASE program, such as a string, variable name, or series of commands.

Description

The `#define` directive defines an identifier and optionally lets you replace text in a program before compilation. Each `#define` definition must begin on a new line and is limited to 4096 characters.

Identifiers are available only to the program in which they are defined. To define identifiers for use in multiple programs, use `#include`. To remove an identifier so that it is no longer used as a preprocessor directive, use `#undef`.

Typically, use the `#define` directive for the following purposes:

- to declare an identifier with no replacement text, so you can use it with the `#ifdef` or `#ifndef` directive.
- to declare an identifier and assign replacement text to represent a constant value or a complex expression, so you can use it with the `#if` directive.
- to create an inline function (pseudo function).

`#define` statements override memory variables, built-in commands and functions, and any other element having the same name as *<identifier>*. This is shown in the following examples.

```
* Overriding a memory variable
#define mValue 10
mValue = 25    && dBASE returns an error
               && because mValue is a constant
mVariable = 25
#define mVariable 10
? mVariable           && displays 10

* Overriding a built-in function
#define upper(x) "? 'bad idea' "    && UPPER( ) is a
                                   && built-in function
? upper("mvar") && returns 'bad idea'
```

For more information about using the `#define` directive, see Chapter 7 in the *Programmer's Guide*.

Declaring identifiers

Defining an identifier without replacement text lets you use it with the `#ifdef` or `#ifndef` directive to test if the identifier exists. This is useful for setting conditions to either include or exclude code for compilation, as in the following example.

```
#define firstrun
...

#ifdef firstrun
    <compile these program setup statements>
    <for example, include debugging commands>
#endif
```

If you later modify your program so you don't need to compile the program setup statements, remove or comment the `#define` statement.

Declaring identifiers to represent constants

Assign an identifier to represent a constant value or expression when you want the preprocessor to search for and replace all instances of the identifier with the specified value or expression before compilation. This is shown in the following example.

```
#define cCompany "dSolutions, Inc."
...
? cCompany      && displays "dSolutions, Inc."
```

This search-and-replace capability, known as *macro expansion*, can streamline your code and improve its readability because you can use a single identifier to represent a frequently used constant or a complex expression. In addition, if you need to change the value of a constant in your program, you need to change only the constant definition and not every occurrence of the constant.

To replace an identifier only in parts of a program, insert `#undef <identifier>` into your program where you want the search-and-replace process to stop.

Creating inline functions

When the preprocessor encounters a function call that matches the function definition, it replaces the function call with the replacement text, inserting the arguments of the function call into the replacement text. This is shown in the following example.

```
#define Avg(num1,num2) (num1+num2)/2
...
nNumber1=20
nNumber2=40
? Avg(nNumber1,nNumber2)    && displays 30
```

Unlike standard dBASE user-defined functions (UDFs), the number of arguments passed from a function call must match the number of parameters defined in your `#define` statement.

Note There are important differences between how dBASE evaluates inline functions and user-defined functions (UDFs). If you don't understand these differences, your inline functions might not return the values you expect. For best results, always enclose the replacement text in parentheses to ensure the proper order of evaluation. For more information, see Chapter 7 in the *Programmer's Guide*.

Nesting preprocessor macros

You can nest preprocessor macros; that is, a macro can expand into another macro. The following example shows how the third macro definition depends on the first two:

```
#define K_CR CHR(13)
#define K_LF CHR(10)
#define K_CRLF K_CR + K_LF
```

Example

The following examples use `#define` to specify font parameters so that a desired font can be called by a single identifier:

```
#define BIGFONT fontname "Roman",fontheight 20,;
fontwidth 22
#define TEXTFONT fontname "Roman",fontheight 11,;
fontwidth 6
#define ENTRYFONT fontname "Roman",fontheight 16,;
fontitalic .t.
```

The following examples use `#define` to specify a function that trims all values passed as parameters and set a predetermined field list.

#if

```
#define ALLTRIM(x) LTRIM(RTRIM(x))
#define FIELDLIST "Flight_no,Origin,Dest,Date,;
    Departure,Arrival,Price"
```

Portability

Not supported in dBASE III PLUS.

See Also

#if, **#ifdef**, **#ifndef**, **#include**, **#undef**

#if

Preprocessor

Controls conditional compilation of code based on the value of an identifier assigned with **#define**.

Syntax

```
#if <condition>
<statements 1>
[#else
<statements 2>]
#endif
```

<condition> A logical expression, using an identifier you've defined, that evaluates to true or false.

<statements 1> One or more program lines consisting of any combination of commands, functions, and preprocessor directives. These lines are compiled if **<condition>** evaluates to true.

#else <statements 2> Specifies the program lines to compile if **<condition>** evaluates to false.

Description

Use the **#if** directive to conditionally compile sections of source code based on the value of **<identifier>**. Two other directives, **#ifdef** and **#ifndef**, are also used to conditionally include or exclude code for compilation. Unlike the **#if** directive, however, they test only for the existence of an identifier, not for its value.

Conditional compilation is useful when maintaining different versions of the same program or for debugging. This is shown in the following example.

```
# define mInvmod = "incomplete"
...
#if mInvmod="complete"
    <Inventory module commands here>
#else
    mMessage = "Inventory not ready to compile"
#endif
```

For more information about using the **#if** directive, see Chapter 7 in the *Programmer's Guide*.

Example

The following example uses `#if/#else` to determine if a preprocessor directive containing a field list has been declared, and if not, uses `#define` to specify a field list:

```
#define FIELDLIST "Company, Contact, Phone"
*
*
*   additional program
*
CLEAR
USE Clients
#if FIELDLIST = "Company, Contact, Phone"
    LIST TO PRINT
#else
    #define FIELDLIST "Company, Contact, Phone"
    LIST TO PRINT
#endif
```

Portability

Not supported in dBASE III PLUS or dBASE IV, but supported in the dBASE Compiler for DOS.

See Also

`#define`, `#ifdef`, `#ifndef`

#ifdef

Preprocessor

Controls conditional compilation of code based on the existence of an identifier assigned with `#define` or the existence of the `__dbasewin__` identifier.

Syntax

```
#ifdef <identifier>
<statements 1>
[#else
<statements 2>]
#endif
```

<identifier> The identifier to test for existence. *<identifier>* is defined with the `#define` directive.

<statements 1> One or more program lines consisting of any combination of commands, functions, and preprocessor directives. These lines are compiled if *<identifier>* has been defined.

#else <statements 2> Specifies the program lines to compile if *<identifier>* has not been defined.

`#ifndef`

Description

Use the `#ifdef` directive to conditionally compile sections of source code. If you've defined *<identifier>* with `#define`, the code you specify with *<statements 1>* is compiled; otherwise, the code following `#else` is compiled.

Conditional compilation is useful when maintaining different versions of the same program or for debugging purposes. This is shown in the following example.

```
#define InvMod
#ifdef InvMod
    <Inventory module commands here>
#else
    mMessage = "Inventory not ready to compile"
#endif
```

To determine whether *Visual* dBASE or a prior version of dBASE is currently running, use `#ifdef` with the `__dbasewin__` identifier, as shown in the following example.

```
#ifdef __dbasewin__
    < code specific to dBASE for Windows>
#else
    <code specific to prior versions of dBASE>
#endif
```

For more information about using the `#ifdef` directive, see Chapter 7 in the *Programmer's Guide*.

Example

The following example uses `#ifdef` to determine if a desired font has been defined prior to printing. If not, the font is defined and printing is initiated.

```
#ifdef BIGFONT
    LIST TO PRINT
#else
    #define BIGFONT fontname "Roman",;
        fontheight 20,fontwidth 22
    LIST TO PRINT
#endif
```

Portability

Not supported in dBASE III PLUS.

See Also

`#define`, `#if`, `#ifndef`

#ifndef

Preprocessor

Controls conditional compilation of code based on the existence of an identifier assigned with `#define`.

Syntax

```
#ifndef <identifier>
<statements 1>
[#else
<statements 2>]
#endif
```

<identifier> The identifier to test for existence. *<identifier>* is defined with the #define directive.

<statements 1> One or more program lines consisting of any combination of commands, functions, and preprocessor directives. These lines are compiled if *<identifier>* has not been defined.

#else <statements 2> Specifies the program lines to compile if *<identifier>* has been defined.

Description

Use the #ifndef directive to conditionally compile sections of source code. If you haven't defined *<identifier>* with #define, the code you specify with *<statements 1>* is compiled; otherwise, the code following #else is compiled.

Conditional compilation is useful when maintaining different versions of the same program or for debugging. This is shown in the following example.

```
#ifndef InvMod
    mMessage = "Inventory not ready to compile"
#else
    <Inventory module commands here>
#endif
```

For more information about using the #ifndef directive, see Chapter 7 in the *Programmer's Guide*.

Example

The following example uses #ifndef to determine if a desired font has not been defined, and if not, defines the font with #define. The #else portion serves to confirm that the font is correctly defined by first undefining and then defining "BIGFONT".

```
#ifndef BIGFONT
    #define BIGFONT fontname "Roman",;
    fontheight 20,fontwidth 22
#else
    #undef BIGFONT
    #define BIGFONT fontname "Roman",;
    fontheight 20,fontwidth 22
#endif
```

Portability

Not supported in dBASE III PLUS.

See Also

#define, #if, #ifdef

Inserts the contents of the specified source file (known as an *include* or *header* file) into the current program file at the location of the `#include` statement.

Syntax

```
#include <filename>| "<filename>"
```

<filename>| "<filename>" The name of the file, optionally including a full or partial path, whose contents are to be inserted into the current program file. You can specify the file name within or without quotes. An include file typically has an `.h` file-name extension.

If you specify `<filename>` without a path, the preprocessor uses the following search order:

- 1 It searches the current directory for the file exactly as you've specified it.
- 2 If you omitted the `.h` file-name extension, it adds the extension and searches the current directory.
- 3 If it can't find the file in the current directory, it looks in `<home directory>\INCLUDE`. (The home directory is the one in the `_dbwinhome` system memory variable.)
- 4 If it can't find the file in the current directory or `<home directory>\INCLUDE`, it looks in the directory you specify in DOS with the `INCLUDE` environment variable.

Description

Identifiers are typically available only to the program in which they are defined. To use a single set of identifiers in multiple programs, save the `#define` statements in a file, then use the `#include` directive to define the identifiers in additional programs. The file containing the `#define` statements is called an include file. For example, an include file named `IDENT.H` might contain the following directives:

```
#define cCompany="dSolutions, Inc."
#define nSales=40000
#define cHomeState="TX"
```

Each program that needs access to these identifiers would contain the following line:

```
#include ident.h
```

An advantage of having all the `#define` statements in one file is the ease of maintenance. If you need to modify any of the `#define` statements, you need only change the include file; the program files that use the `#define` statements remain unchanged. After you modify the include file, recompile your program file for the changes to take effect.

For example, if you later need to change the home state value in the previous example from `"TX"` to `"WA"` throughout your application, you need to change the `#define` statement only in `IDENT.H`.

For more information about include files, see Chapter 7 in the *Programmer's Guide*.

Example

The following example inserts the contents of a user defined setup file in the currently running dBASE program. The file "setup.h" can contain #define identifier definitions that are common to several programs the user runs:

```
#include "c:\dbasewin\setup\setup.h"
```

Portability

Not supported in dBASE III PLUS or dBASE IV, but supported in the dBASE Compiler for DOS.

See Also

#define, GETFILE()

#pragma

Preprocessor

Sets compiler options.

Syntax

```
#pragma <coverage(on | off)>
```

#pragma <coverage(on | off)> Includes or excludes coverage analysis, which provides information about which program lines are executed.

Description

Use #pragma coverage(on) in your program to start coverage analysis each time you run the program, instead of executing SET COVERAGE ON in the Command window before compiling and running the program.

This is also the only way to start coverage analysis from within a program; if you issue SET COVERAGE ON in a program, dBASE doesn't create or update a coverage file for that program. For more information about coverage files, see SET COVERAGE.

For more information about preprocessor directives, see Chapter 7 in the *Programmer's Guide*.

Example

The following example uses #pragma to set coverage on for a debugging session:

```
#ifdef DEBUG
    #pragma coverage(on)
#else
    #pragma coverage(off)
#endif
```

Portability

Not supported in dBASE IV or dBASE III PLUS.

See Also

#define, SET COVERAGE

#undef

Preprocessor

Removes the current definition of the specified identifier previously defined with #define.

Syntax

#undef <identifier>

<identifier> The identifier whose definition is to be removed.

Description

The #undef directive removes the definition of an identifier previously defined with the #define directive. If you use #define with <replacement text>, the preprocessor replaces all instances of the identifier with the replacement text from the point it encounters that #define until it encounters an #undef specifying the same identifier. Therefore, to replace an identifier only in parts of a program, insert #undef <identifier> into your program where you want the search-and-replace process to stop.

Example

See #ifndef for an example of using #undef.

Portability

Not supported in dBASE III PLUS.

See Also

#define



Classes

Classes

CLASS ARRAY

An object that stores multiple memory variables in locations you can access individually.

Properties

The following table lists the properties of the Array class. For more information on each property, see Chapter 8.

Property	Default	Description
Add()	N/A	Adds elements to a one-dimensional array object
ClassName	ARRAY	Identifies the array class
Count()	N/A	Returns the number of elements in the associated array
Delete()	N/A	Deletes an element from a one-dimensional array object, or deletes a row or column from a two-dimensional array object
Dimensions	1	Specifies the number of dimensions in the array object
Dir()	N/A	Stores the name, size, date stamp, time stamp, and DOS attribute(s) of files to an array object
DirExt()	N/A	Same as Dir(), but with extra columns for Windows 95 file information
Element()	N/A	Returns the number of a specified element in the array object
Fields()	N/A	Stores structure information of the current table to the array object
Fill()	N/A	Inserts a specified value into one or more locations in an array object
Grow()	N/A	Adds elements to the array object
Insert()	N/A	Inserts an element into a one-dimensional array object, or inserts a row or column of elements into a two-dimensional array object
ReleaseAll()	N/A	Deletes all elements of the associated array.
ReleaseKey()	N/A	Deletes a specified element from the associated array
Resize()	N/A	Increases or decreases the number of elements in the array object
Scan()	N/A	Searches an array object for a specified value

Property	Default	Description
Size	N/A	Contains the number of elements in an array object
Sort()	N/A	Sorts the elements in a one-dimensional array object, or sorts rows in a two-dimensional array object
Subscript()	N/A	Returns the row number or the column number of a specified element in an array object

Description

Use an array object to store multiple related items in memory. For more information on using array objects, see Chapter 10 in the *Programmer's Guide*.

When you create an array object with the NEW operator, you can specify two parameters:

- *<rows expN>*—The number of elements if the array object is one-dimensional, or the number of rows if the array object is two-dimensional.
- *<columns expN>*—The number of elements in each row. You specify *<columns expN>* only if the array object is two-dimensional.

For example, the following command creates a two-dimensional array object with ten rows and twenty elements in each row:

```
MyArray = NEW ARRAY(10, 20)
```

Note You can't use DEFINE to create an array object; instead, use the NEW operator. The NEW operator creates an array object in the following example:

```
Array1 = NEW ARRAY(10, 20)    && Creates array object.  
DEFINE ARRAY Array1          && Returns an error
```

You can also create an array object with the DECLARE command:

```
DECLARE Array1[10,20]    && Creates an array object Array1
```

Arrays can contain other arrays as elements:

```
Array1 = NEW ARRAY(10, 20)    && Creates array object.  
Array1[5,5] = NEW ARRAY(2,2) && Places array in 5th row, 5th column  
Array1[5,5][1,1] = 10        && Places value in 1st row, 1st column  
                                && of the array element
```

You can also create a *literal array* object by specifying the values you want to place in the array. Like other arrays, literal arrays can contain other arrays as elements. Literal arrays are useful for populating items such as list boxes and tab boxes.

```
LitArray1 = {2,4,6}  
? LitArray1[2]                && Returns 4  
LitArray2 = {"Jan", "Feb"}  
? LitArray2[1]                && Returns "Jan"  
z={1, NEW ARRAY(2,2),3}  
? z[2]                        && Returns Array  
? z[2][1,2]                   && Returns .F.
```

Example

The following example creates a form with one button for each field in a table. Field names are stored in an array object named AButtons:

```

LOCAL loFields
loFields = NEW FieldForm()
loFields.OPEN()
CLASS FieldForm OF FORM
    USE ?
    * create a button for each field
    AButtons = NEW ARRAY(1)
    FOR i = 1 TO FCOUNT()
        AButtons.ADD(1)
        AButtons[i] = NEW PUSHBUTTON(THIS)
        AButtons[i].text = FIELD(i)
        AButtons[i].top = i * 2
        AButtons[i].left = 5
        AButtons[i].width = 20
    NEXT
ENDCLASS

```

See Also

CLASS ASSOCARRAY, CLASS LISTBOX, CLASS TABBOX, DECLARE, PUBLIC

CLASS ASSOCARRAY

An array object class that takes character strings as subscripts.

Properties

The following table lists the properties of the Assocarray class. For more information on each property, see Chapter 8.

Property	Default	Description
ClassName	ASSOCARRAY	Identifies the assocarray class
Count()	N/A	Returns the number of elements in the associated array
FirstIndex	N/A	Returns the subscript character string for an element of an associated array
IsIndex()	N/A	Returns .T. if the specified character expression is a subscript of the associated array
NextIndex()	N/A	Returns the subscript of the next element in the associated array
RemoveAll()	N/A	Deletes all elements of the associated array.
RemoveKey()	N/A	Deletes a specified element from the associated array

Description

Use the ASSOCARRAY class to create an array that has character strings as subscripts. This lets you assign meaningful information to both the subscript and the array element it references. For more information on using array objects, see ARRAYXREF.

Use the standard array operator [] to add and reference items in the array. An empty string "" can be used as a subscript.

Example

The following example creates an associated array and displays its subscripts and contents. It then deletes a specified element from the array.

```
aa = NEW ASSOCARRAY()
aa["San Francisco"] = "49ers"
aa["Los Angeles"] = "Rams"
x = aa.FirstIndex
DO WHILE .NOT. EMPTY(x)
    ? x, aa[x]           && display element subscript and contents
    x = aa.NextIndex(x)  && 'increments' index pointer
ENDDO
? aa.Count()    && Returns 2
aa.RemoveKey("San Francisco")&& Removes element from the array
? aa.Count()    && Returns 1
```

See Also

CLASS ARRAY, DECLARE, PUBLIC

CLASS BROWSE

A data-editing tool that displays multiple records in row-and-column format, or single records in columnar or edit format.

Properties

The following table lists the properties of the Browse class. For more information on each property, see Chapter 8.

Property	Default	Description
Alias	Empty string	Determines the table file that is accessed
Append	.T.	Determines if records can be added
Before	N/A	Specifies which object the browse object precedes in the tabbing order of the parent form
ClassName	BROWSE	Identifies the browse object's class
ColorHighlight	WindowText /Window	Sets the color of the browse object when it's highlighted
ColorNormal	BtnText/ BtnFace	Sets the color of the browse object
CUATab	.T.	Determines cursor behavior when you press <i>Tab</i>
Copy()	N/A	Copies selected text to the Windows clipboard
Cut()	N/A	Cuts selected text and places it on the Windows clipboard
Delete	.T.	Determines if records can be marked for deletion
Enabled	.T.	Determines if the browse object can be selected

Property	Default	Description
Fields	Empty string	Specifies the fields to display, and the field options to apply to each field
FieldWidth	N/A	Specifies the width of a character field in a browse object
Follow	.T.	Determines if the display follows a record to its new index order when a key field value is changed
FontBold	.F.	Determines if characters are displayed in bold type
FontItalic	.F.	Determines if characters are displayed in italic type
FontName	MS Sans Serif	Specifies the font to apply to displayed characters
FontSize	N/A	Specifies the size of the font in points
Header3D	.T.	Specifies whether the top portion of the browse object appears raised (three-dimensional)
Height	N/A	Sets the height
HelpFile	Empty string	Identifies a Windows Help file (.HLP) that contains context-sensitive Help topics
HelpID	Empty string	Specifies the context string or context number of a Help topic in a Windows Help file (.HLP)
hWnd	N/A	Returns the browse object handle
ID	-1	Identifies the browse object with a numeric value
Keyboard()	N/A	Passes a character string to the browse object, simulating typed user input
Left	N/A	Sets the position of the left border
Mode	0	Specifies the display format
Modify	.T.	Determines if the user can alter records
MousePointer	0	Specifies the mouse pointer type when the pointer is over the browse object
Move()	N/A	Moves or sizes the browse object
Name	BROWSE1	Specifies the browse object's name
OnAppend	N/A	Executes a subroutine when a record is added to the table
OnChange	N/A	Executes a subroutine when the user changes a value
OnGotFocus	N/A	Executes a subroutine when the browse object receives focus
OnHelp	N/A	Executes a subroutine when the user presses <i>F1</i>
OnLeftDblClick	N/A	Executes a subroutine when the user double-clicks the browse object
OnLeftMouseDown	N/A	Executes a subroutine when the user clicks the browse object with the left mouse button
OnLeftMouseUp	N/A	Executes a subroutine when the user releases the left mouse button while the pointer is over the browse object
OnLostFocus	N/A	Executes a subroutine when focus is removed
OnMiddleDblClick	N/A	Executes a subroutine when the user double-clicks the browse object with the middle mouse button
OnMiddleMouseDown	N/A	Executes a subroutine when the user clicks the browse object with the middle mouse button
OnMiddleMouseUp	N/A	Executes a subroutine when the user releases the middle mouse button while the pointer is on the browse object
OnMouseMove	N/A	Executes a subroutine when the user moves the mouse over the browse object

Property	Default	Description
OnNavigate	N/A	Executes a subroutine when the user moves to a different record
OnOpen	N/A	Executes a subroutine when the parent form is opened
OnRightDbClick	N/A	Executes a subroutine when the user double-clicks the browse object with the right mouse button
OnRightMouseDown	N/A	Executes a subroutine when the user clicks the browse object with the right mouse button
OnRightMouseUp	N/A	Executes a subroutine when the user releases the right mouse button while the pointer is on the browse object
PageNo	N/A	Specifies on which page of a multi-page form the browse object appears
Parent	N/A	An object reference that points to the parent form
Paste()	N/A	Copies text from the Windows clipboard to the current cursor position
Release()	N/A	Removes the browse object definition from memory
ScrollBar	2 (Auto)	Determines if the browse object has a scroll bar
SetFocus()	N/A	Gives focus to the browse object
ShowDeleted	.T.	Determines if the browse object's delete box column is displayed
ShowHeading	.T.	Determines if field name headings are displayed at the top of each column in the browse object
ShowRecNo	.T.	Determines if the browse object's record number column is displayed
StatusMessage	Empty string	Specifies a message to display on the status bar while the browse object has focus
TabStop	.T.	Determines if the user can give object focus to the browse object by pressing <i>Tab</i> or <i>Shift+Tab</i>
Text	N/A	Specifies a character string to display on the caption bar
Toggle	.T.	Determines if the user can switch between display modes
Top	N/A	Sets the position of the top border
Undo()	N/A	Reverses the effects of the most recent Cut(), Copy(), or Paste() action
Visible	.T.	Determines whether the browse object is visible or hidden
When	.T.	Specifies a condition that must evaluate to true before the user can give focus to the browse object
Width	N/A	Sets the width

Description

Use a browse object to allow viewing and editing of records from a table. A browse object offers most the capabilities and options of the BROWSE and EDIT commands.

To display data one record at a time (single-record format), set the Mode property to 1 or 2. Setting Mode to 1 specifies form layout, and setting Mode to 2 specifies columnar layout. To display data in multiple-record format, set Mode to 0. When you set the Toggle property to true, the user can switch between all three formats by pressing F2.

Two properties specify which table is displayed in the browse object.

- The View property of the parent form

- The Alias property of the browse object

The View property bases the form on a query (QBE), which is automatically invoked when you open the form. The Alias property determines which table opened by the query is displayed. An *alias* is an alternate name given to an open table file, and can consist of.

- A name you specify with the ALIAS option of the USE command.
- The table file name (if you did not assign the table an alias).
- The letter that corresponds to the work area of the table.
- The number, preceded by an underscore character (_), that corresponds to the work area of the table.

A browse object is often used to display child records in a multi-table form. For example, when the parent form is based on a query that opens two or more files in a parent-child relation, you can specify the child table with the Alias property. For more information on aliases and work areas, see Alias, SELECT, and USE.

You can specify individual fields to display with the Fields property. For example, if the browse object's form is based on a query, you use Fields to display fields from any of the query's tables. (You must specify a file with Alias before you can use Fields.)

You determine a browse object's dimensions with the FROM...TO clause of its DEFINE command or with its Height and Width properties.

When you create a browse object with the NEW operator, you can specify two parameters:

- *<parent form reference>*—An object reference pointing to the parent form.
- *<object name expC>*—A character string assigned to the Name property of the new browse object. This value is optional.

For example, the following commands create a form and a browse object to display in it:

```
MyForm = NEW FORM()
MyBrow = NEW BROWSE(MyForm, "OurBrow")
```

The Name property of the new browse object contains "OurBrow".

Note You can specify a value for the View property with the Choose View dialog box, which lets you choose query or a table. To access the Choose View dialog box, click on the Tool Button next to the View item in the Inspector.

You can specify an Alias value with the Choose Alias dialog box, which lists all open tables. To access the Choose Alias dialog box, click on the Tool Button next to the Alias item in the Inspector.

You can specify a Fields value with the Choose Field dialog box, which lists fields from all open tables. To access the Choose Field dialog box, click on the Tool Button next to the Fields item in the Inspector.

If you use the Inspector, don't enclose the View, Alias, or Fields value with quotation marks. If you do, dBASE displays an error message.

Example

The following example defines a form that contains a browse object of the Contact table. The Exit pushbutton gives the user an alternative way to close the form:

```

LOCAL f
f=NEW EntryForm()
f.OPEN()
CLASS EntryForm OF FORM
  this.Top=2
  this.Left=2
  this.Width=38
  this.Height=20
  this.View = "Contact.DBF"
  this.Text= "Edit as Required"
  DEFINE BROWSE Br1 OF THIS;
  PROPERTY;
    Alias "Contact",;
    Fields "CompCode, Contact",;
    Top 4,,;
    Left 3,,;
    Width 32,,;
    Height 12,,;
    Delete .T.,;
    StatusMessage "Contact Table Browse",;
    Toggle .F.  && Disables switching views with F2
  DEFINE TEXT Text1 OF THIS;
  PROPERTY;
    Text "Contact Table Points of Contact",;
    Width 38,,;
    Top 1,,;
    Left 0,,;
    Alignment 1,,;
    Height 2.50,,;
    FontSize 12.00,,;
    FontBold .T.
  DEFINE PUSHBUTTON Exit OF THIS;
  PROPERTY Text "Exit", Height 2,,;
    Top 17, Left 14,,;
    OnClick {;Form.Close()}
ENDCLASS

```

See Also

DEFINE, BROWSE, EDIT, OPEN FORM, RELEASE OBJECT

CLASS CHECKBOX

An object that lets users toggle logical values between true (.T.) and false (.F.).

Properties

The following table lists the properties of the Checkbox class. For more information on each property, see Chapter 8.

Property	Default	Description
Before	N/A	Specifies which object the check box precedes in the tabbing order of the parent form
ClassName	CHECKBOX	Identifies the check box class
ColorNormal	BtnText/ BtnFace	Sets the color of the check box
DataLink	Empty string	Links the check box to a field
Enabled	.T.	Determines if the check box can be selected
FontBold	.T.	Determines if characters in the check box prompt are displayed in bold type
FontItalic	.F.	Determines if characters are displayed in italic type
FontName	MS Sans Serif	Specifies the font to apply to displayed characters
FontSize	N/A	Specifies the size of the font in points
FontStrikeOut	.F.	Determines if characters are displayed in strikeout type
FontUnderline	.F.	Determines if characters are displayed in underlined type
Group	.T.	Starts an object group in the parent form
Height	N/A	Sets the height
HelpFile	Empty string	Identifies a Windows Help file (.HLP) that contains context-sensitive Help topics
HelpID	Empty string	Specifies the context string or context number of a Help topic in a Windows Help file (.HLP)
hWnd	N/A	Specifies the check box handle
ID	-1	Identifies the check box with a numeric value
Left	N/A	Sets the position of the left border
MousePointer	0	Specifies the mouse pointer type when the pointer is over the check box
Move()	N/A	Moves or sizes the check box
Name	CHECKBOX1	Specifies the check box name
OldStyle	.F.	Determines if the check box is displayed in the default Windows style or in dBase style
OnChange	N/A	Executes a subroutine when the user toggles between values
OnGotFocus	N/A	Executes a subroutine when the check box receives focus
OnHelp	N/A	Executes a subroutine when the user presses <i>F1</i>
OnLeftDblClick	N/A	Executes a subroutine when the user double-clicks the check box
OnLeftMouseDown	N/A	Executes a subroutine when the user clicks the check box with the left mouse button
OnLeftMouseUp	N/A	Executes a subroutine when the user releases the left mouse button while the pointer is over the check box
OnLostFocus	N/A	Executes a subroutine when focus is removed
OnMiddleDblClick	N/A	Executes a subroutine when the user double-clicks the check box with the middle mouse button
OnMiddleMouseDown	N/A	Executes a subroutine when the user clicks the check box with the middle mouse button

Property	Default	Description
OnMiddleMouseUp	N/A	Executes a subroutine when the user releases the middle mouse button while the pointer is on the check box
OnMouseMove	N/A	Executes a subroutine when the user moves the mouse pointer over the check box
OnOpen	N/A	Executes a subroutine when the parent form is opened
OnRightDbClick	N/A	Executes a subroutine when the user double-clicks the check box with the right mouse button
OnRightMouseDown	N/A	Executes a subroutine when the user clicks the check box with the right mouse button
OnRightMouseUp	N/A	Executes a subroutine when the user releases the right mouse button while the pointer is on the check box
PageNo	N/A	Specifies on which page of a multi-page form the check box object appears
Parent	N/A	An object reference that points to the parent form
Release()	N/A	Removes the check box definition from memory
SetFocus()	N/A	Gives focus to the check box
SpeedTip	Empty string	Specifies the text that appears when the mouse remains on the check box for more than one second
StatusMessage	Empty string	Specifies a message to display on the status bar while the check box has focus
TabStop	.T.	Determines if the user can give focus to the check box by pressing <i>Tab</i> or <i>Shift+Tab</i>
Text	N/A	Specifies a character string to display next to the check box
Top	0	Sets the position of the top border
Value	.F.	Determines whether the check box is selected (contains an x) or is blank
Visible	.T.	Determines whether the check box is visible or hidden
When	N/A	Specifies a condition that must evaluate to true before the user can give focus to the check box
Width	10.25	Sets the width

Description

Use a check box to let users toggle a value between true and false, yes and no, or on and off. This value can be contained in a logical field that you specify with the *DataLink* property.

The presence of an X in a check box means true, yes, or on, and the absence of an X means false, no, or off.

When you link a check box to a field and the user moves from record to record, the check box setting changes to reflect each value.

When you create a check box with the *NEW* operator, you can specify two parameters:

- *<parent form reference>*—An object reference pointing to the parent form.
- *<object name expC>*—A character string assigned to the *Name* property of the new check box. This value is optional.

For example, the following commands create a form and a check box to display in it:

```
MyForm = NEW FORM()
MyCBox = NEW CHECKBOX(MyForm, "OurCBox")
```

The Name property of the new check box contains "OurCBox".

Note You can specify a field for the DataLink property with the Field Picker. To access the Field Picker, click on the Tool Button next to the DataLink item in the Inspector.

Example

The following example uses check boxes for selecting options of the LIST command. BigBox is a subclass of Checkbox with a big font:

```
USE ?
DEFINE FORM ListForm PROPERTY MDI .F.
DEFINE PUSHBUTTON Lister OF ListForm ;
    PROPERTY ;
    Text "List", ;
    Top 10, ;
    Left 15, ;
    OnClick ListClick
NEW bigBox(ListForm,"Include record numbers")
NEW bigBox(ListForm,"Send to printer")
ListForm.Checkbox1.Top = 2
ListForm.Checkbox2.Top = 6
READMODAL(ListForm)

CLASS bigBox(f,lcText) OF CHECKBOX(f)
    this.FontName = "Arial"
    this.FontSize = 16
    this.Text      = SPACE(2) + lcText
    this.Width     = LEN(lcText) * 2
    this.Left      = 2
    this.Value     = .F.
ENDCLASS

FUNCTION ListClick
    IF Form.Checkbox1.Value    && include record;
                                numbers
        IF Form.Checkbox2.Value && to printer
            LIST TO PRINTER
        ELSE
            LIST
        ENDIF
    ELSE
                                && no record numbers
        IF Form.Checkbox2.Value && to printer
            LIST OFF TO PRINTER
        ELSE
            LIST OFF
        ENDIF
    ENDIF
    FORM.CLOSE()
RETURN 0
```

See Also

CLASS RADIOBUTTON, DEFINE

CLASS COMBOBOX

An object that lets users select a value from a list by entering characters in a text box or by selecting the value directly from the list.

Properties

The following table lists the properties of the Combobox class. For more information on each property, see Chapter 8.

Property	Default	Description
Before	N/A	Specifies which object the combo box precedes in the tabbing order of the parent form
ClassName	COMBOBOX	Identifies the combo box class
ColorNormal	WindowText/ Window	Sets the color of the combo box
Copy()	N/A	Copies selected text to the Windows clipboard
Cut()	N/A	Cuts selected text and places it on the Windows clipboard
DataLink	Empty string	Links the combo box to a field
DataSource	Empty string	Specifies the prompts to display in the combo box
DropDownHeight	6	Specifies the number of lines displayed in the list portion of the combo box
Enabled	.T.	Determines if the combo box can be selected
FontBold	.T.	Determines if characters are displayed in bold type
FontItalic	.F.	Determines if characters are displayed in italic type
FontName	MS Sans Serif	Specifies the font to apply to displayed characters
FontSize	N/A	Specifies the size of the font in points
FontStrikeOut	.F.	Determines if characters are displayed in strikeout type
FontUnderline	.F.	Determines if characters are displayed in underlined type
Height	N/A	Sets the height
HelpFile	Empty string	Identifies a Windows Help file (.HLP) that contains context-sensitive Help topics
HelpID	Empty string	Specifies the context string or context number of a Help topic in a Windows Help file (.HLP)
hWnd	N/A	Returns the combo box handle
ID	-1	Identifies the combo box with a numeric value
Keyboard()	N/A	Passes a character string to the combo box, simulating typed user input
Left	N/A	Sets the position of the left border
MousePointer	0	Specifies the mouse pointer type when the pointer is over the combo box
Move()	N/A	Moves or sizes the combo box
Name	COMBOBOX1	Specifies the combo box name
OnChange	N/A	Executes a subroutine when the user moves from one prompt to another
OnGotFocus	N/A	Executes a subroutine when the combo box receives focus
OnHelp	N/A	Executes a subroutine when the user presses F1

Property	Default	Description
OnLeftDblClick	N/A	Executes a subroutine when the user double-clicks the combo box
OnLeftMouseDown	N/A	Executes a subroutine when the user clicks the combo box with the left mouse button
OnLeftMouseUp	N/A	Executes a subroutine when the user releases the left mouse button while the pointer is over the combo box
OnLostFocus	N/A	Executes a subroutine when focus is removed
OnMiddleDblClick	N/A	Executes a subroutine when the user double-clicks the combo box with the middle mouse button
OnMiddleMouseDown	N/A	Executes a subroutine when the user clicks the combo box with the middle mouse button
OnMiddleMouseUp	N/A	Executes a subroutine when the user releases the middle mouse button while the pointer is on the combo box
OnMouseMove	N/A	Executes a subroutine when the user moves the mouse in the combo box
OnOpen	N/A	Executes a subroutine when the parent form is opened
OnRightDblClick	N/A	Executes a subroutine when the user double-clicks the combo box with the right mouse button
OnRightMouseDown	N/A	Executes a subroutine when the user clicks the combo box with the right mouse button
OnRightMouseUp	N/A	Executes a subroutine when the user releases the right mouse button while the pointer is on the combo box
PageNo	N/A	Specifies on which page of a multi-page form the combo box object appears
Parent	N/A	An object reference that points to the parent form
Paste()	N/A	Copies text from the Windows clipboard to the combo box
Release()	N/A	Removes the combo box definition from memory
SetFocus()	N/A	Gives focus to the combo box
Sorted	.F.	Determines if the prompts are listed in sorted order
StatusMessage	Empty string	Specifies a message to display on the status bar while a combo box has focus
Style	1	Determines how a combo box appears, and how the user selects or inputs values
TabStop	.T.	Determines if the user can give focus to the combo box by pressing <i>Tab</i> or <i>Shift+Tab</i>
Top	N/A	Sets the position of the top border
Undo()	N/A	Reverses the effects of the most recent Cut(), Copy() or Paste() action
Value	N/A	Sets the value in the combo box
Visible	.T.	Determines whether the combo box is visible or hidden
When	N/A	Specifies a condition that must evaluate to true before the user can give the combo box focus
Width	N/A	Sets the width

Description

Use a combo box to display a list of options and allow users to search through the options rapidly.

For example, combo box prompts might consist of last names from a field in an indexed table. The user could locate the first record containing a particular last name by entering the name's initial characters in the text box. As the user enters the characters, the prompt scrolls to the top of the display. Alternatively, the user could use the scroll bars to visually locate the last name, then select it directly with the mouse.

A combo box can display five different types of prompt:

- 1 File names
- 2 Values in a table field
- 3 Field names from the structure of a table
- 4 Elements in an array object
- 5 The names of all tables in the currently open database (See OPEN DATABASE for information on databases.)

Specify the prompts with the DataSource property.

Determine the dimensions of a combo box with the FROM...TO clause of the DEFINE command or with the Height and Width properties. If the height you specify isn't enough for all the prompts, the user can scroll through the prompts. If the height you specify is greater than needed, dBASE reduces the height automatically.

When you create a combo box with the NEW operator, you can specify two parameters:

- *<parent form reference>*—An object reference pointing to the parent form.
- *<object name expC>*—A character string assigned to the Name property of the new combo box. This value is optional.

For example, the following commands create a form and a combo box to display in it:

```
MyForm = NEW FORM()
MyComb = NEW COMBOBOX(MyForm, "OurComb")
```

The Name property of the new combo box contains "OurComb".

Notes You can specify a DataSource value with the Choose Data Source dialog box in the Form Designer. To access the Choose Data Source dialog box, click the Tool Button next to the DataSource item in the Inspector.

You can specify a field for the DataLink property with the Choose Field dialog box. To access the Choose Field dialog box, click the Tool Button next to the DataLink item in the Inspector.

Example

The following example defines a form that contains a combo box that displays names from the Animals.DBF table. Double clicking on one of the choices closes the form:

```
LOCAL f
f=NEW Showbox()
f.OPEN()
CLASS Showbox OF FORM
  this.View = "ANIMALS.DBF"
```

```

DEFINE COMBOBOX CB1 OF THIS;
PROPERTY;
    DataSource "FIELD Animals->Name",;
    FontBold .T.,;
    Top 4,,
    Left 6,,
    Width 20, ;
    OnLeftDblClick {; ? 'You picked...';
        THIS.value, ; FORM.CLOSE()}
DEFINE TEXT Text1 OF THIS;
PROPERTY;
    Text "Pick Your Favorite Animal",;
    FontBold .T., Width 40,,
    Top 1, Left 3
ENDCLASS

```

See Also

CLASS LISTBOX, DEFINE

CLASS DDELINK

Initiates and controls a DDE link between dBASE and a server application, allowing dBASE to send instructions and data-exchange requests to the server.

Properties

The following table lists the properties of the DDELink class. For more information on each property, see Chapter 8.

Property	Default	Description
Advise()	N/A	Requests that the server notify the client when an item in the server document changes
ClassName	DDELINK	Identifies the DDELink class
Execute()	N/A	Sends instructions to the server in its own language
Initiate()	N/A	Starts a conversation with a DDE server application
OnNewValue	N/A	Executes a subroutine when an item in the server application changes
Peek()	N/A	Retrieves a data item stored in a server document
Poke()	N/A	Inserts a data item into a server document
Reconnect()	N/A	Restores a DDE link that was terminated with Terminate()
Release()	N/A	Removes the DDELink object definition from memory
Server	N/A	Contains the name of the server you specified with the Initiate() method
Terminate()	N/A	Terminates the link with the server application
TimeOut	1000	Determines the amount of time dBASE waits for a transaction before returning an error
Topic	Empty string	Contains the name of the topic you specified with the Initiate() method
Unadvise()	N/A	Asks the server to stop notifying the DDELink object when an item in the server document changes

Description

Use a DDELink object to open a channel of communication (known as a *DDE link*) between dBASE and an external Windows application (known as a *server*). You can exchange data and instructions through this link, making the two applications work together. For example, a DDElink object might establish a link to Quattro Pro for Windows, open one of its notebook files, and copy its data into a dBASE table.

Establish a DDE link with the Initiate() property. If a session in the server application is not already running, Initiate() tries to start a session before establishing the link. If the attempt is unsuccessful, Initiate() returns a value of false.

For more information on DDE, see Chapter 26 in the *Programmer's Guide*. For information on using dBASE as a server application, see CLASS DDETOPIC.

Example

The following example creates a DDELINK object with the NEW operator, attempts to initiate a server session with Quattro Pro for Windows with the Initiate method, uses the Peek method to extract a value from element GasCosts:B5 to a dBASE memory variable and Poke to place a new value in cell GasCosts:B6 of the QPW spreadsheet:

```
LOCAL LinkObj
LinkObj = NEW DDELINK()
IF LinkObj.Initiate("QPW", "Tutor.WB1")
    ? "Connection to QPW initiated"
ELSE
    ? "Connection to QPW failed"
ENDIF
mValue1=LinkObj.Peek("GasCosts:B5")
? mValue1
LinkObj.Poke("GasCosts:B6", "198")
mValue2=LinkObj.Peek("GasCosts:B6")
? mValue2
LinkObj.RELEASE()
```

See Also

CLASS DDETOPIC, CLASS OLE, CLASS OLEAUTOCLIENT, DEFINE

CLASS DDETOPIC

Determines the actions taken when *Visual* dBASE receives requests from a DDE client.

Properties

The following table lists the properties of the DDETopic class. For more information on each property, see Chapter 8.

Property	Default	Description
ClassName	DDETOPIC	Identifies the DDETopic class
Notify()	N/A	Notifies all client applications that a dBASE item was changed
OnAdvise	N/A	Executes a subroutine when an external application creates a hot link

Property	Default	Description
OnExecute	N/A	Executes a subroutine when a client application sends a directive to dBASE
OnPeek	N/A	Executes a subroutine when the client requests a value from dBASE
OnPoke	N/A	Executes a subroutine when the client inserts a new value into a dBASE item
OnUnadvise	N/A	Executes a subroutine when a client removes a hot link from a particular item
Release()	N/A	Removes the DDETopic object definition from memory
Topic	N/A	The DDETopic object's topic

Description

Use a DDETopic object to determine what dBASE does for a client application when dBASE is the server in a DDE link.

As a server application, *Visual* dBASE accepts directives from a client application, accepts and sends data items to the server application, and performs whatever actions you specify with DDETopic object properties. For example, an OnPoke subroutine might receive a field name and a field value from a client application, insert the value into the designated field, and notify the client application with the Notify() method.

You usually create a DDETopic object in an initiation-handler routine, which you assign to the OnInitiate property of _app. An initiation-handler executes when a client application requests a DDE link with dBASE. For more information on initiation handlers, see Chapter 26 in the *Programmer's Guide*.

For information on using dBASE as a client application, see CLASS DDELINK.

When you create a DDETopic object with the NEW operator, specify the <topic> parameter. This value is automatically placed in the Topic property of the new DDETopic object. External applications use this property to identify the object and establish a DDE link. For example, the following command creates a DDETopic object, and puts "MyTopic" in the Topic property:

```
OurTopic = NEW DDETopic("MyTopic")
```

A Quattro Pro spreadsheet might execute the following command to invoke the new object:

```
{INITIATE "DBASEWIN", "MyTopic"}
```

Example

The following example creates a DDE server for handling stock information. For simplicity, only one hot link is supported at a time, and all stocks have the same value. When a client buys a stock (Execute feature), the stock price goes up; when a client sells a stock, the stock price goes down.

```
SET PROCEDURE TO PROGRAM(1) ADDITIVE
PUBLIC Stock, Adviser, Value
Value = 100.0
_app.DDEServiceName = "Stock"
_app.OnInitiate = INITHANDLER
```

CLASS DDETOPIC

```
FUNCTION InitHandler
PARAMETER Topic
IF ".STK" $ Topic
    x = NEW StockTopic()
ELSE
    x = .F.
ENDIF
RETURN x

CLASS StockTopic OF DDETOPIC
Stock.OnAdvise = AdvHandler
Stock.OnExecute = ExeHandler
Stock.OnPeek = PeekHandler
Stock.OnPoke = PokeHandler
Stock.OnUnadvise = UnAdvHandler

FUNCTION AdvHandler
PARAMETER Item
Adviser = Item
RETURN .T.

FUNCTION ExeHandler
PARAMETER Cmd
IF Cmd = "SELL"
    Value = Value - 10.0
ELSE
    IF Cmd = "BUY"
        Value = Value + 10.0
    ENDIF
ENDIF
THIS.Notify(Adviser)
RETURN .T.

FUNCTION PeekHandler
PARAMETER Item
RETURN Value

FUNCTION PokeHandler
PARAM Item, Val
? "POKE: ", Item, Val
RETURN .T.

FUNCTION UnAdvHandler
PARAMETER Item
IF(Adviser = item)
    Adviser = .F.
ENDIF
RETURN .T.

ENDCLASS
```

See Also

CLASS DDELINK

CLASS EDITOR

A tool that lets the user view and edit a text file or memo field.

Properties

The following table lists the properties of the Editor class. For more information on each property, see Chapter 8.

Property	Default	Description
Before	N/A	Specifies which object the editor object precedes in the tabbing order of the parent form
Border	.F.	Determines if the editor object is surrounded with a border
ClassName	EDITOR	Identifies the editor class
ColorNormal	WindowText/ Window	Sets the color of the editor object when it does not have focus
Copy()	N/A	Copies selected text to the Windows clipboard
Cut()	N/A	Cuts selected text and places it on the Windows clipboard
CUATab	.T.	Determines cursor behavior when you press <i>Tab</i>
DataLink	Empty string	Links the editor object to a text file, a memo field, or a character field.
Enabled	.T.	Determines if the editor object can be selected
FontBold	.T.	Determines if characters are displayed in bold type
FontItalic	.F.	Determines if characters are displayed in italic type
FontName	MS Sans Serif	Specifies the font to apply to characters in the display
FontSize	N/A	Specifies the size of the font in point size
FontStrikeOut	.F.	Determines if characters are displayed in strikeout type
FontUnderline	.F.	Determines if characters are displayed in underlined type
Height	N/A	Sets the height
HelpFile	Empty string	Identifies a Windows Help file (.HLP) that contains context-sensitive help topics
HelpID	Empty string	Specifies the context string or context number of a help topic in a Windows Help file (.HLP)
hWnd	N/A	Returns the editor object handle
ID	-1	Identifies the editor object with a numeric value
Keyboard()	N/A	Passes a character string to the editor object, simulating typed user input
Left	N/A	Sets the position of the left border
LineNo	1	Sets the current line in the editor object
Modify	.T.	Determines if the user can alter data in the editor object
MousePointer	0	Specifies the mouse pointer type when the pointer is over the editor object
Move()	N/A	Moves or sizes the editor object
Name	EDITOR1	Specifies the name of the editor object
OnChange	N/A	Executes a subroutine when the user changes text in the editor object
OnGotFocus	N/A	Executes a subroutine when the editor object receives focus

Property	Default	Description
OnHelp	N/A	Executes a subroutine when the user presses F1
OnLeftDbtClick	N/A	Executes a subroutine when the user double-clicks on the editor object
OnLeftMouseDown	N/A	Executes a subroutine when the user clicks on the editor object with the left mouse button
OnLeftMouseUp	N/A	Executes a subroutine when the user releases the left mouse button while the pointer is over the editor object
OnLostFocus	N/A	Executes a subroutine when focus is removed
OnMiddleDbtClick	N/A	Executes a subroutine when the user double-clicks on the editor object with the middle mouse button
OnMiddleMouseDown	N/A	Executes a subroutine when the user clicks on the editor object with the middle mouse button
OnMiddleMouseUp	N/A	Executes a subroutine when the user releases the middle mouse button while the pointer is on the editor object
OnMouseMove	N/A	Executes a subroutine when the user moves the mouse in the editor object
OnOpen	N/A	Executes a subroutine when the parent form is opened
OnRightDbtClick	N/A	Executes a subroutine when the user double-clicks on the editor object with the right mouse button
OnRightMouseDown	N/A	Executes a subroutine when the user clicks on the editor object with the right mouse button
OnRightMouseUp	N/A	Executes a subroutine when the user releases the right mouse button while the pointer is on the editor object
PageNo	N/A	Specifies on which page of a multi-page form the editor object appears
Parent	N/A	An object reference that points to the parent form
Paste()	N/A	Copies text from the Windows clipboard to the current cursor position
Release()	N/A	Removes the editor object definition from memory
Scrollbar	1 (On)	Determines if the editor object has a scroll bar
SetFocus()	N/A	Gives focus to the editor object
StatusMessage	Empty string	Specifies a message to display on the status bar while the editor object has focus
TabStop	.T.	Determines if the user can give focus to the editor object by pressing Tab or Shift+Tab
Top	N/A	Sets the position of the top border
Undo()	N/A	Reverses the effect of the most recent Cut(), Copy() or Paste() action
Valid	N/A	Specifies a condition that must evaluate to true (.T.) before the user can remove focus from the editor object
Value	Empty string	Determines the contents of the editor object
Visible	.T.	Determines whether the editor object is visible or hidden
When	N/A	Specifies a condition that must evaluate to true before the user can give focus to the editor object
Width	N/A	Sets the width
Wrap	.T.	Determines if the editor object wraps input text automatically

Description

Use an editor object to give a form text-editing capability, letting users view and change the contents of text files, memo fields, and character fields.

Use the DataLink property to specify the text file or memo field to access. To access a text file, use the keyword FILE, as with:

```
"FILE MYTEXT.TXT"
```

To access a memo field or a character field, use the field name and the alias of the table that contains the field.

To set the dimensions of the editor object, use the FROM...TO clause of its DEFINE command or with its Height and Width properties. The scroll bars let the user move horizontally or vertically within the display when input exceeds the dimensions of the editor object.

To position the editor object on the form, use the Top and Left properties.

When you create an editor object with the NEW operator, you can specify two parameters:

- *<parent form reference>*—An object reference pointing to the parent form.
- *<object name expC>*—A character string assigned to the Name property of the new editor object. This value is optional.

For example, the following commands create a form and an editor object to display in it:

```
MyForm = NEW FORM()
MyEdit = NEW EDITOR(MyForm, "OurEdit")
```

The Name property of the new editor object contains "OurEdit".

Example

The following example defines a form that contains a browse object containing CompCode and Contact information from the Contact table plus the associated memo field contents displayed in an editor object to the right. Back and Next pushbuttons advance or retard the record pointer while Exit gives the user an alternative way to close the form:

```
LOCAL f
f=NEW EntryForm()
f.OPEN()
CLASS EntryForm OF FORM
  this.Top=2
  this.Left=2
  this.Width=72
  this.Height=20
  this.View = "Contact.DBF"
  this.Text= "Edit as Required"
  DEFINE BROWSE Br1 OF THIS;
  PROPERTY;
  Top 4,;
  Left 3,;
  Width 32,;
```

CLASS ENTRYFIELD

```

    Height 12
    DEFINE TEXT Text1 OF THIS;
    PROPERTY;
    Text "Contact Table Points of Contact",;
    Width 72,;
    Top 1,;
    Left 0,;
    Alignment 1,;
    Height 2.50,;
    FontBold .T.,;
    FontSize 14.00,;
    ColorNormal "R/W"
    DEFINE EDITOR ED1 OF THIS;
    PROPERTY;
    Top 4,;
    Left 37,;
    Width 32,;
    Height 12,;
    DataLink "Contact->Notes"
    DEFINE PUSHBUTTON Back OF THIS;
    PROPERTY Text "Back", Height 2,;
    Top 17, Left 22,;
    OnClick {;SKIP-1}, FontBold .T.
    DEFINE PUSHBUTTON Next OF THIS;
    PROPERTY TEXT "Next", Height 2,;
    Top 17, Left 32,;
    OnClick {;SKIP}, FontBold .T.
    DEFINE PUSHBUTTON Exit OF THIS;
    PROPERTY Text "Exit", Height 2,;
    Top 17, Left 42,;
    OnClick {;Form.Close()}, FontBold .T.
ENDCLASS
```

See Also
MODIFY COMMAND

CLASS ENTRYFIELD

An area in which the user can input or modify a single value.

Properties

The following table lists the properties of the Entryfield class. For more information on each property, see Chapter 8.

Property	Default	Description
Before	N/A	Specifies which object the entry field precedes in the tabbing order of the parent form
Border	.T.	Determines if the entry field is surrounded with a border
ClassName	ENTRYFIELD	Identifies the Entryfield class

Property	Default	Description
ColorHighLight	WindowText/ Window	Sets the color of the entry field when it is highlighted
ColorNormal	WindowText/ Window	Sets the color of the entry field when it isn't highlighted
Copy()	N/A	Copies selected text to the Windows clipboard
Cut()	N/A	Cuts selected text and places it on the Windows clipboard
DataLink	Empty string	Links the entry field to a table field
Enabled	.T.	Determines if the entry field can be selected
FontBold	.T.	Determines if characters in the entry field are displayed in bold type
FontItalic	.F.	Determines if characters are displayed in italic type
FontName	MS Sans Serif	Specifies the font to apply to displayed characters
FontSize	N/A	Specifies the size of the font in points
FontStrikeOut	.F.	Determines if characters are displayed in strikeout type
FontUnderline	.F.	Determines if characters are displayed in underlined type
Function	Empty string	Formats displayed text
Height	N/A	Sets the height
HelpFile	Empty string	Identifies a Windows Help file (.HLP) that contains context-sensitive Help topics
HelpID	Empty string	Specifies the context string or context number of a Help topic in a Windows Help file (.HLP)
hWnd	N/A	Returns the entry field handle
ID	-1	Identifies the entry field with a numeric value
Key	N/A	Executes a subroutine when the user presses a key
Keyboard()	N/A	Passes a character string to the entry field, simulating typed user input
Left	N/A	Sets the position of the left border
MaxLength	N/A	Specifies the scrolling width
MousePointer	0	Specifies the mouse pointer type when the pointer is over the entry field
Move()	N/A	Moves or sizes the entry field
Name	ENTRYFIELD1	Specifies the name of the entry field
OldStyle	.F.	Determines if the entry field is displayed in the default Windows style or in dBase style
OnChange	N/A	Executes a subroutine when the user changes a value
OnGotFocus	N/A	Executes a subroutine when the entry field receives focus
OnHelp	N/A	Executes a subroutine when the user presses <i>F1</i>
OnLeftDblClick	N/A	Executes a subroutine when the user double-clicks the entry field
OnLeftMouseDown	N/A	Executes a subroutine when the user clicks the entry field with the left mouse button
OnLeftMouseUp	N/A	Executes a subroutine when the user releases the left mouse button while the pointer is over the entry field
OnLostFocus	N/A	Executes a subroutine when focus is removed
OnMiddleDblClick	N/A	Executes a subroutine when the user double-clicks the entry field with the middle mouse button

Property	Default	Description
OnMiddleMouseDown	N/A	Executes a subroutine when the user clicks the entry field with the middle mouse button
OnMiddleMouseUp	N/A	Executes a subroutine when the user releases the middle mouse button while the pointer is on the list box
OnMouseMove	N/A	Executes a subroutine when the user moves the mouse over the entry field
OnOpen	N/A	Executes a subroutine when the parent form is opened
OnRightDbClick	N/A	Executes a subroutine when the user double-clicks the entry field with the right mouse button
OnRightMouseDown	N/A	Executes a subroutine when the user clicks the entry field with the right mouse button
OnRightMouseUp	N/A	Executes a subroutine when the user releases the right mouse button while the pointer is on the entry field
PageNo	N/A	Specifies on which page of a multi-page form the entry field object appears
Parent	N/A	An object reference that points to the parent form
Paste()	N/A	Copies text from the Windows clipboard to the current cursor position
Picture	Empty string	Formats text
Release()	N/A	Removes the entry field definition from memory
SelectAll	.T.	Determines if the initial value in the entry field appears selected (highlighted) when the entry field receives focus
SetFocus()	N/A	Gives focus to the entry field
SpeedTip	Empty string	Specifies the text that appears when the mouse remains on the entry field for more than one second
StatusMessage	Empty string	Specifies a message to display on the status bar while the entry field has focus
TabStop	.T.	Determines if the user can give focus to the entry field by pressing <i>Tab</i> or <i>Shift+Tab</i>
Top	N/A	Sets the position of the top border
Undo()	N/A	Reverses the effect of the most recent Cut(), Copy() or Paste() action
Valid	N/A	Specifies a condition that must evaluate to true (.T.) before the user can remove focus from the entry field
ValidErrorMsg	Invalid input	Specifies a character string to display on the status bar when the Valid property returns false (N/A)
ValidRequired	.F.	Determines if the Valid property applies to all data or to new data only
Value	EntryField	Sets the value in the entry field
Visible	.T.	Determines whether the entry field is visible or hidden
When	N/A	Specifies a condition that must evaluate to true before the user can give focus to the entry field
Width	N/A	Sets the width

Description

Use an entry field to let users enter and edit data in a single field. For example, a form that allows access to only one table field might use an entry field instead of a browse object.

An entry field physically resembles an entry area created by the dBASE IV command @...GET. However, entering data in an entry field doesn't require a READ command.

Use the DataLink property to specify the field to edit. If you wish, you can specify a field for the DataLink property with the Field Picker in the Form Designer. To access the Field Picker, click on the Tool Button next to the DataLink item in the Inspector.

Create entry field prompts with text objects. For example, a text object saying "Enter password" might appear above an entry field that accepts passwords. For information on text objects, see CLASS TEXT.

When you create an entry field with the NEW operator, you can specify two parameters:

- *<parent form reference>*—An object reference pointing to the parent form.
- *<object name expC>*—A character string assigned to the Name property of the new entry field. This value is optional.

For example, the following commands create a form and an entry field to display in it:

```
MyForm = NEW FORM()
MyField = NEW ENTRYFIELD(MyForm, "OurField")
```

The Name property of the new entry field contains "OurField".

Example

The following example defines a form that contains three entry fields. Each ENTRYFIELD uses a different object-oriented definition syntax, all of which work in Visual dBASE. The Back and Next pushbuttons let the user move the record pointer:

```
LOCAL F1
F1=NEW EntryForm()
F1.OPEN()
CLASS EntryForm OF FORM
  this.View="Company.DBF"
  this.Top=2
  this.Left=2
  this.Width=38
  this.Height=13
  this.Text="Edit as Required"
  * NEW operator syntax:
    CompCode=NEW ENTRYFIELD(this)
    CompCode.DataLink="Company->CompCode"
    CompCode.Top=5
    CompCode.Left=2
    CompCode.Width=5
    CompCode.Height=1.5
  * Combination syntax:
    DEFINE ENTRYFIELD Type OF THIS
      this.Type.Width=5
      this.Type.Top=5
      this.Type.Left=9
      this.Type.Height=1.5
      this.Type.DataLink="Company->Type"
  * DEFINE object syntax
    DEFINE ENTRYFIELD Company OF THIS;
```

```
PROPERTY;
    Width 20,;
    Top 5,;
    Left 16,;
    Height 1.5,;
    DataLink "Company->Company"
DEFINE TEXT Text1 OF THIS;
PROPERTY;
    Text "Company Information",;
    Width 34,;
    Top 1,;
    Left 2,;
    Alignment 7,;
    Height 2.50,;
    FontSize 12.00,;
    Border .T.
DEFINE PUSHBUTTON Back OF THIS;
PROPERTY Text "Back", Height 2,;
    Top 10, Left 10,;
    OnClick {;SKIP-1}
DEFINE PUSHBUTTON Next OF THIS;
PROPERTY TEXT "Next", Height 2,;
    Top 10, Left 20,;
    OnClick {;SKIP}
ENDCLASS
```

See Also
@...SAY...GET, DEFINE

CLASS FORM

A customized window containing objects for input and output.

Properties

The following table lists the properties of the Form class. For more information on each property, see Chapter 8.

Property	Default	Description
AbandonRecord()	N/A	Releases from memory a record created with BeginAppend()
ActiveControl	N/A	Contains a reference to the object that currently has focus
AutoSize	.F.	Determines if the form adjusts itself automatically to contain its objects when the form is opened
BeginAppend()	N/A	Creates a temporary buffer in memory for a record that is based on the structure of the current table
CanClose	N/A	Executes a subroutine that determines if a form can be closed when an attempt is made to close the form
ClassName	FORM	Identifies the Form class
Close()	N/A	Closes the form
ColorNormal	BtnText/ BtnFace	Sets the color of the form

Property	Default	Description
Enabled	.T.	Determines if the form can receive focus
EscExit	.T.	Determines if the user can close the form by pressing <i>Esc</i>
First	N/A	Contains an object reference that points to the first object in the form
Height	N/A	Sets the height
HelpFile	Empty string	Identifies a Windows Help file (.HLP) that contains context-sensitive Help topics
HelpID	Empty string	Specifies the context string or context number of a Help topic in a Windows Help file (.HLP)
hWnd	N/A	Returns the object handle of the form
Icon	N/A	Specifies an icon format file (.ICO) or resource that displays when a form is minimized
IsRecordChanged()	.F.	Returns a logical value that indicates whether the current record in the append buffer, created with <code>BeginAppend()</code> , has been modified
Left	N/A	Sets the position of the left border
Maximize	.T.	Determines if the form can be maximized
MDI	.T.	Determines if the form conforms to Windows Multiple Document Interface (MDI) standards
MenuFile	Empty string	Assigns a predefined menu system to the form
Minimize	.T.	Determines if the form can be minimized
MousePointer	0	Specifies the mouse pointer type when the pointer is over the form
Move()	N/A	Specifies the mouse pointer type when the pointer is over the form
Moveable	.T.	Determines if the form can be moved with the mouse
NextCol()	N/A	The next highest column position
NextObj	N/A	Returns a reference to the next object in the form's tabbing order
NextRow()	N/A	The next highest row position
OnAppend	N/A	Executes a subroutine when a record is added to a table on which the form is based
OnChange	N/A	Executes a subroutine when the user changes a value in an object
OnClose	N/A	Executes a subroutine when the form is closed
OnGotFocus	N/A	Executes a subroutine when the form receives focus
OnHelp	N/A	Executes a subroutine when the user presses <i>F1</i>
OnLeftDbtClick	N/A	Executes a subroutine when the user double-clicks the form
OnLeftMouseDown	N/A	Executes a subroutine when the user clicks the form with the left mouse button
OnLeftMouseUp	N/A	Executes a subroutine when the user releases the left mouse button while the pointer is over the form
OnLostFocus	N/A	Executes a subroutine when focus is removed
OnMiddleDbtClick	N/A	Executes a subroutine when the user double-clicks the form with the middle mouse button
OnMiddleMouseDown	N/A	Executes a subroutine when the user clicks the form with the middle mouse button
OnMiddleMouseUp	N/A	Executes a subroutine when the user releases the middle mouse button while the pointer is on the form

Property	Default	Description
OnMouseMove	N/A	Executes a subroutine when the user moves the mouse in the form
OnMove	N/A	Executes a subroutine after the user moves the form
OnNavigate	N/A	Executes a subroutine when the user moves to a different record
OnOpen	N/A	Executes a subroutine when the form is opened
OnRightDbClick	N/A	Executes a subroutine when the user double-clicks the form with the right mouse button
OnRightMouseDown	N/A	Executes a subroutine when the user clicks the form with the right mouse button
OnRightMouseUp	N/A	Executes a subroutine when the user releases the right mouse button while the pointer is on the form
OnSelection	N/A	Executes a subroutine when the user submits the form
OnSize	N/A	Executes a subroutine after the user resizes the form
Open()	N/A	Opens the form as a modeless window
PageCount()	N/A	Returns the highest numbered page defined for the form
PageNo	N/A	Specifies the active page of the form
PopupMenu	N/A	Specifies a Popup menu for the form
Print()	N/A	Prints the form and the objects it contains
ReadModal()	N/A	Opens the form as a modal window
Refresh()	N/A	Updates data displayed in control objects within a form
Release()	N/A	Removes the form definition from memory
SaveRecord()	N/A	Saves to the current table a record created with BeginAppend()
ScaleFontName	MS Sans Serif	Determines which font the coordinate plane of the form is based on
ScaleFontSize	N/A	Determines the height of each row and the width of each column in the coordinate plane of the form
Scrollbar	0 (Off)	Determines if the form has a scroll bar
SetFocus()	N/A	Gives focus to the form
ShowSpeedTip	.T.	Determines if hint balloons appear for controls on a form when the mouse rests on those controls
Sizeable	.T.	Determines if the user can resize the form
StatusMessage	Empty string	Specifies a message to display on the status bar while a form has focus
StatusMessage	Empty string	Specifies a message to display on the status bar while the form has focus
SysMenu	.T.	Determines if the form has a Control menu
Text	N/A	Specifies a character string to display in the caption bar
Top	N/A	Sets the position of the top border
TopMost	.F.	Specifies whether modal forms display on top of all other forms
View	Empty string	Specifies the query or table on which the form is based
Visible	.T.	Determines whether the form is visible or hidden
Width	N/A	Sets the width
WindowState	0 (Normal)	Determines whether the form is minimized, maximized, or normal

Description

A form is a window you design. It can contain standard Windows interface objects (also called *controls*) that let users enter, access, and modify data. For example, a form might display a browse object (for editing records), entry fields (for entering single values) and check boxes (for setting a logical field to true or false). A form is a *container* for the objects it displays. Consequently, releasing a form definition from memory automatically releases the definitions of the objects it contains.

A form can consist of more than one page. One way to implement multi-page forms is to use the PageNo property of controls to determine on which page they appear, and use a set of tabs to let users easily switch between pages. For more information, see CLASS TABBOX.

Most forms are based on a query (QBE), which you specify with the View property. For example, to base a form on a QBE named CONTACT.QBE, set the View property to "CONTACT.QBE". Then use the DataLink and DataSource properties to link the objects contained by the form to fields in the QBE tables.

You can create two types of forms: *modal* and *modeless*. In the Windows environment, a modal form window is like a dBASE IV window. A modal form halts execution of the routine that opened it until the form is closed. When active, it takes control of the user interface; users can't switch to another window without exiting the form. A dialog box is an example of a modal form; when it is opened, program execution stops and focus can't be given to another window until the user closes the dialog box.

In contrast a modeless form window allows users to freely switch to other windows in an application. Most forms that you create for a Windows application will typically be modeless. A modeless form window conforms to the Multiple Document Interface (MDI) protocol, which lets you open multiple document windows within an application window.

To create and use a modeless form, set the MDI property to true (.T.) and open the form with the Open() method or the OPEN FORM command. To create and use a modal form, set MDI to false (.F.) and open the form with the ReadModal() method or the READMODAL() function.

You can also create form windows that appear like application windows. To do so, set the MDI property to false and use SHELL(.F.). SHELL(.F.) hides the standard dBASE environment and lets your form take over the user interface. The dBASE application window disappears, and the form name appears in the Windows Task List.

When you create an form with the NEW operator, you can specify a value for the Text property with the <text> parameter. For example, the following commands create a form and an editor object:

```
MyForm = NEW FORM("This is my form.")
```

When this form is opened, "This is my form" is displayed in the caption bar.

Example

The following example uses DEFINE FORM, NEW FORM() and a subclass of form to create similar forms:

```
DEFINE FORM Doggy;
  PROPERTY;
  Text "Oly",;
  Top 5,;
  Left 5
OPEN FORM Doggy

Kitty = NEW FORM()
Kitty.Text = "Lacey"
Kitty.Top  = 10
Kitty.Left = 10
Kitty.OPEN()

Buddy = NEW Toni()
Buddy.OPEN()
CLASS Toni OF FORM
  this.Text = "Toni"
  this.Top  = 15
  this.Left = 15
ENDCLASS
```

See Also

CLASS TABBOX, DEFINE, MDI, Open(), OPEN FORM, ReadModal(), READMODAL(), SHELL()

CLASS IMAGE

An area that displays bitmap images.

Properties

The following table lists the properties of the Image class. For more information on each property, see Chapter 8.

Property	Default	Description
Alignment	0	Positions a graphic image in the image object
Before	N/A	Specifies which object the image object precedes in the tabbing order of the parent form
ClassName	IMAGE	Identifies the image class
DataSource	Empty string	Specifies which DLL resource, file, or binary field that contains the graphic image
Height	N/A	Sets the height
hWnd	N/A	Returns the image object handle
ID	-1	Identifies the image object with a numeric value
Left	N/A	Sets the position of the left border

Property	Default	Description
MousePointer	0	Specifies the mouse pointer's type when the pointer is over the image object
Move()	N/A	Moves or sizes the image object
Name	IMAGE1	Specifies the name of the image object
OnLeftDbClick	N/A	Executes a subroutine when the user double-clicks the image object
OnLeftMouseDown	N/A	Executes a subroutine when the user clicks the image object with the left mouse button
OnLeftMouseUp	N/A	Executes a subroutine when the user releases the left mouse button while the pointer is over the image object
OnMiddleDbClick	N/A	Executes a subroutine when the user double-clicks the image object with the middle mouse button
OnMiddleMouseDown	N/A	Executes a subroutine when the user clicks the image object with the middle mouse button
OnMiddleMouseUp	N/A	Executes a subroutine when the user releases the middle mouse button while the pointer is on the image object
OnMouseMove	N/A	Executes a subroutine when the user moves the mouse in the image object
OnOpen	N/A	Executes a subroutine when the parent form is opened
OnRightDbClick	N/A	Executes a subroutine when the user double-clicks the image object with the right mouse button
OnRightMouseDown	N/A	Executes a subroutine when the user clicks the image object with the right mouse button
OnRightMouseUp	N/A	Executes a subroutine when the user releases the right mouse button while the pointer is on the image object
Parent	N/A	An object reference that points to the parent form
PageNo	N/A	Specifies on which page of a multi-page form the image object appears
Release()	N/A	Removes the image object definition from memory
Top	N/A	Sets the position of the top border
Visible	.T.	Determines whether the image object is visible or hidden
Width	N/A	Sets the width

Description

Use an image object to display images in a form. For example, an image object might display bitmap pictures stored in a binary field.

You can link an image object to any of three sources with the DataSource property:

- A file containing a bitmap image (.BMP or .PCX).
- A binary field containing bitmap images.
- A bitmap resource in a DLL file.

When you link an image object with a binary field and the user moves from record to record, the image stored in each record is displayed.

The image you display in an image object is read-only.

When you create an image object with the NEW operator, you can specify two parameters:

- *<parent form reference>*—An object reference pointing to the parent form.
- *<object name expC>*—A character string assigned to the Name property of the new image object. This value is optional.

For example, the following commands create a form and an image object to display in it:

```
MyForm = NEW FORM()  
MyImage = NEW IMAGE(MyForm, "OurImage")
```

The Name property of the new image object contains "OurImage".

Note You can select a bitmap image with the Choose Bitmap dialog box. To access the Choose Bitmap dialog box, click on the Tool Button next to the DataSource item in the Inspector.

Example

The following example creates a subclass of Form containing a subclass of Image. The subclass of Image places a bitmap in the form:

```
LOCAL ShowPlane  
ShowPlane = NEW PlaneForm()  
ShowPlane.OPEN()  
  
CLASS PlaneForm OF FORM  
  NEW PlaneBMP(this)  
ENDCLASS  
  
CLASS PlaneBmp(form) OF IMAGE(form)  
  this.DataSource = "FILE Airbrlnd.BMP"  
ENDCLASS
```

See CLASS LISTBOX for an additional example of using CLASS IMAGE.

See Also

CLASS SHAPE, DEFINE, RESTORE IMAGE

CLASS LINE

A line you display at a specified location in a form.

Properties

The following table lists the properties of the Line class. For more information on each property, see Chapter 8.

Property	Default	Description
Before	N/A	Specifies which object the line object precedes in the tabbing order of the parent form
Bottom	N/A	Sets the row position of the lower end of the line object
ClassName	LINE	Identifies the line class
ColorNormal	BtnText/ BtnFace	Sets the color of the line object

Property	Default	Description
Left	N/A	Sets the horizontal position of the left end of the line object
Name	LINE1	Specifies the name of the line object
OnOpen	N/A	Executes a subroutine when the parent form is opened
PageNo	N/A	Specifies on which page of a multi-page form the line object appears
Parent	N/A	An object reference that points to the parent form
Pen	0 (Solid)	Specifies the line object's style
Release()	N/A	Removes the line object definition from memory
Right	N/A	Sets the column position right end of the line object
Top	N/A	Sets the row position of the higher end of the line object
Visible	.T.	Determines whether the line object is visible or hidden
Width	1	Sets the thickness of the line object

Description

Use a line object to draw a line on a form. A line object can underline another object or mark a boundary between two areas in a form.

The user can't give focus to a line object.

When you create a line object with the NEW operator, you can specify two parameters:

- *<parent form reference>*—An object reference pointing to the parent form.
- *<object name expC>*—A character string assigned to the Name property of the new line object. This value is optional.

For example, the following commands create a form and a line object to display in it:

```
MyForm = NEW FORM()
MyLine = NEW LINE(MyForm, "OurLine")
```

The Name property of the new line object contains "OurLine".

Example

The following example uses DEFINE LINE, within a Class definition, to create two magenta lines (one vertical and one horizontal) on the form:

```
LOCAL f
f=NEW EntryForm()
f.OPEN()
CLASS EntryForm OF FORM
    this.Top=2
    this.Left=2
    this.Width=36
    this.Height=13
    this.Text= "Class Line Demo"
*
* other object definitions
*
    DEFINE LINE Ln1 OF THIS;
        PROPERTY;
        Left 10,;
```

CLASS LISTBOX

```
Top 3,;
Width 4,;
Bottom 8,;
ColorNormal "RB"
DEFINE LINE Ln2 OF THIS;
PROPERTY;
Left 3,;
Top 8,;
Width 4,;
Bottom 8,;
Right 33,;
ColorNormal "RB"
ENDCLASS
```

See Also
CLASS RECTANGLE, DEFINE

CLASS LISTBOX

An object that lets users select one or many values from a list.

Properties

The following table lists the properties of the Listbox class. For more information on each property, see Chapter 8.

Property	Default	Description
Before	N/A	Specifies which object the list box precedes in the tabbing order of the parent form
ClassName	LISTBOX	Identifies the list box class
ColorHighLight	WindowText /Window	Sets the color of the list box when it's highlighted
ColorNormal	WindowText /Window	Sets the color of the list box when it isn't highlighted
Count()	N/A	Returns the number of prompts in the list box
CurSel	0	Specifies the currently-selected prompt in the list box
DataSource	Empty string	Determines which data is displayed in the list box
Enabled	.T.	Determines if the list box can be selected
FontBold	.T.	Determines if characters are displayed in bold type
FontItalic	.F.	Determines if characters are displayed in italic type
FontName	MS Sans Serif	Specifies the font to apply to displayed characters
FontSize	N/A	Specifies the size of the font in point size
FontStrikeOut	.F.	Determines if characters are displayed in strikeout type
FontUnderline	.F.	Determines if characters are displayed in underlined type
Height	N/A	Sets the height
HelpFile	Empty string	Identifies a Windows Help file (.HLP) that contains context-sensitive Help topics

Property	Default	Description
HelpID	Empty string	Specifies the context string or context number of a Help topic in a Windows Help file (.HLP)
hWnd	N/A	Returns the list box handle
ID	-1	Identifies the list box with a numeric value
Left	N/A	Sets the position of the left border
MousePointer	0	Specifies the mouse pointer type when the pointer is over the list box
Move()	N/A	Moves or sizes the list box
Multiple	.F.	Determines if more than one item in the list box can be selected
Name	LISTBOX1	Specifies the name of the list box
OldStyle	.F.	Determines if the list box is displayed in the default Windows style or in dBase style
OnGotFocus	N/A	Executes a subroutine when the list box receives focus
OnHelp	N/A	Executes a subroutine when the user presses <i>F1</i>
OnLeftDblClick	N/A	Executes a subroutine when the user double-clicks the list box
OnLeftMouseDown	N/A	Executes a subroutine when the user clicks the list box with the left mouse button
OnLeftMouseUp	N/A	Executes a subroutine when the user releases the left mouse button while the pointer is over the list box
OnLostFocus	N/A	Executes a subroutine when focus is removed
OnMiddleDblClick	N/A	Executes a subroutine when the user double-clicks the list box with the middle mouse button
OnMiddleMouseDown	N/A	Executes a subroutine when the user clicks the list box with the middle mouse button
OnMiddleMouseUp	N/A	Executes a subroutine when the user releases the middle mouse button while the pointer is on the list box
OnMouseMove	N/A	Executes a subroutine when the user moves the mouse in the list box
OnOpen	N/A	Executes a subroutine when the parent form is opened
OnRightDblClick	N/A	Executes a subroutine when the user double-clicks the list box with the right mouse button
OnRightMouseDown	N/A	Executes a subroutine when the user clicks the list box with the right mouse button
OnRightMouseUp	N/A	Executes a subroutine when the user releases the right mouse button while the pointer is on the list box
OnSelChange	N/A	Executes a subroutine when the highlight is moved from one prompt to another
PageNo	N/A	Specifies on which page of a multi-page form the listbox object appears
Parent	N/A	An object reference that points to the parent form
Release()	N/A	Removes the list box definition from memory
Selected()	N/A	Returns the currently-selected prompt
SetFocus()	N/A	Gives focus to the list box
Sorted	.F.	Determines whether the list box prompts are in sorted order or in natural order
StatusMessage	Empty string	Specifies a message to display on the status bar while the list box has focus

Property	Default	Description
TabStop	.T.	Determines if the user can give focus to the list box by pressing <i>Tab</i> or <i>Shift+Tab</i>
Top	N/A	Sets the position of the top border
Value	Empty string	The currently selected prompt in the list box
Visible	.T.	Determines whether the list box is visible or hidden
When	N/A	Specifies a condition that must evaluate to true before the user can give focus to the list box
Width	N/A	Sets the width

Description

Use a list box to let the user select from a series of prompts. For example, an application might display file names in a list box, letting the user select one or more for deletion.

Specify list box prompts with the `DataSource` property. You can create five different types of prompts:

- 1 File names and subdirectories.
- 2 The contents of a table field.
- 3 Field names from a table.
- 4 Elements in an array object.
- 5 The names of all tables in the currently open database. (See `OPEN DATABASE` for information on databases.)

Determine the dimensions of a list box with the `FROM...TO` clause of the `DEFINE` command or with its `Height` and `Width` properties. If the height you specify isn't enough for all the prompts, the user can scroll through the prompts. If the height you specify is greater than needed, `dBASE` reduces the height automatically.

Pressing the first character of a prompt selects that prompt. If more than one prompt begins with the same character, pressing the character again selects the next prompt that begins with the character.

To let the user choose any number of prompts (or none at all), set the `Multiple` property to true. Each chosen prompt is tagged with a checkmark, and the list box is said to be *multiple-choice*.

There are two ways to determine which prompt or prompts were chosen by the user:

- Check the contents of the `Value` property. `Value` contains the current selection in a single-choice list box or the most recent selection in a multiple-choice list box.
- Use the `LISTSELECTED()` and `LISTCOUNT()` functions in a `DO...WHILE` loop to evaluate the prompts. For more information, see the descriptions of `LISTSELECTED()` and `LISTCOUNT()`.

When you create a list box with the `NEW` operator, you can specify two parameters:

- *<parent form reference>*—An object reference pointing to the parent form.

- *<object name expC>*—A character string assigned to the Name property of the new list box. This value is optional.

For example, the following commands create a form and a list box to display in it:

```
MyForm = NEW FORM()
MyList = NEW LISTBOX(MyForm, "OurList")
```

The Name property of the new list box contains "OurList".

Example

The following example defines a form that contains a list box that displays names from the Animals.DBF table and an image object that displays the associated animal's .BMP image:

```
LOCAL ShowPics
ShowPics=NEW PickForm()
ShowPics.OPEN()
CLASS PickForm OF FORM
  this.View="Animals.DBF"
  this.Top=2
  this.Left=2
  this.Width=60
  this.Height=20
  this.Text= "Animals of the World"
  DEFINE LISTBOX LB1 OF THIS;
  PROPERTY;
    DataSource "FIELD Animals->Name",;
    Top 4,;
    Left 6,;
    Width 20,;
    Height 12
  DEFINE TEXT Text1 OF THIS;
  PROPERTY;
    Text "Pick Your Favorite Animal",;
    FontBold .T.,;
    Width 40,;
    Top 1,;
    Left 3,;
    Height 2.50,;
    FontSize 12.00,;
    ColorNormal "RB/W"
  DEFINE IMAGE Img1 OF THIS;
  PROPERTY;
    DataSource "BINARY Animals->BMP",;
    Top 2,;
    Left 32,;
    Width 25,;
    Height 15
ENDCLASS
```

See Also

DO...WHILE, DEFINE, LISTCOUNT(), LISTSELECTED(), ON SELECTION FORM, OPEN FORM

CLASS MENU

A Windows-style menu system assigned to a form.

Properties

The following table lists the properties of the Menu class. For more information on each property, see Chapter 8.

Property	Default	Description
Before	N/A	Specifies which other menu object the menu object precedes
Checked	.F.	Determines if a checkmark appears beside a menu prompt
ClassName	MENU	Identifies the menu class
Enabled	.T.	Determines if the menu can be selected
HelpFile	Empty string	Identifies a Windows Help file (.HLP) that contains context-sensitive Help topics
HelpID	Empty string	Specifies the context string or context number of a Help topic in a Windows Help file (.HLP)
ID	-1	Identifies the menu with a numeric value
Name	MENU1	Specifies the name of the menu item
OnClick	N/A	Executes a subroutine when the user chooses the menu
OnHelp	N/A	Executes a subroutine when the user presses <i>F1</i>
Parent	Empty string	An object reference that points to the parent form
Release()	N/A	Removes the menu object definition from memory
Separator	.F.	Determines if the menu prompt is a menu item that the user can't select
Shortcut	Empty string	Specifies a key combination that executes the OnClick subroutine
StatusMessage	Empty string	Specifies a message to display on the status bar
Text	N/A	Specifies a character string to display in the menu prompt

Description

Use menu objects to create a menu system for a form.

A menu system consists of two elements:

- The object reference variable that identifies the entire menu system. You must create this variable before you can create the menu system. (The variable name is not displayed anywhere.)
- Menu items, the prompts offered by the menu system. You can display menu items in four places:
 - The dBASE application menu bar, a row near the top of the application frame window. (This happens only when the MDI property is set to true.)
 - The Menu Bar, an unmarked row at the top of the form. (This happens only when the MDI property is set to false.) The first menu item you create is automatically displayed at the left end of the menu bar.

- A pull-down menu, which the user opens by selecting a menu item from the menu bar.
- A cascading menu, which the user opens by selecting a menu item from a pull-down menu or another cascading menu.

You create the object reference variable and menu items with the `DEFINE MENU` command. To create a top-level menu that contains the standard Windows Edit and Window pull-down menus, use `DEFINE MENUBAR`. For more information, see `CLASS MENUBAR`.

You can also add popup menus to a form. Popup menus let users perform an action (usually right-click) to bring up a list of menu items. For more information, see `CLASS POPUP`.

The following commands create a form and declare a name for its menu system:

```
MyForm = NEW FORM()
DEFINE MENU Main OF MyForm
```

The following command creates a menu item (File) on the application menu bar:

```
DEFINE MENU File OF MyForm.Main PROPERTY Text "File"
```

The following command moves the menu item (File) to the menu bar of the form:

```
MyForm.MDI = .F.
```

The following commands create two menu items (Open and Close) in a pull-down menu:

```
DEFINE MENU xOpen OF;
    MyForm.Main.File PROPERTY Text "Open"
DEFINE MENU xClose OF;
    MyForm.Main.File PROPERTY Text "Close"
```

The user opens this pull-down menu by selecting File, the menu item on the menu bar.

The following commands create a cascading menu with two menu items (Close Without Saving, and Close and Save):

```
DEFINE MENU NoSave OF MyForm.Main.File.xClose;
    PROPERTY Text "Close Without Saving"
DEFINE MENU YesSave OF MyForm.Main.File.xClose;
    PROPERTY Text "Close and Save"
```

The user opens this cascading menu by selecting the Close menu item.

Creating pick characters

To let the user select a menu item with a key press, specify a pick character by placing an ampersand to the left of the character in the menu item prompt. For example, the previous command could have specified "S" as the pick character for the Close and Save menu item:

```
DEFINE MENU YesSave OF MyForm.Main.File.xClose;
    PROPERTY Text "Close and &Save"
```

The method for entering a pick character depends on the level of the menu item. When the pick character selects a menu item from the menu bar, the user presses the <Alt> key before entering the character. To select any other menu item, the user inputs the character only.

Assigning actions to menu items

You assign an action to a menu item with the `OnClick` subroutine. For example, the following command assigns a subroutine named `ClsSave` to the `Close` and `Save` menu item:

```
MyForm.Main.File.xClose.YesSave.OnClick = ClsSave
```

Notes You can move a menu object from one form to another by changing the `Parent` property of the menu object. The `Parent` property is read-only for all other classes.

You can design a menu with the `Menu Designer`, a tool that creates a menu file (.MNU). The menu file contains dBASE code that generates the menu you design. To access the `Menu Designer`, click the `Tool` button next to the `MenuFile` property in the `Inspector`.

Example

The following example defines a main menu with two pull-down options, `File` and `Equipment`. `File` has an `Exit` option and `Equipment` has a `Select Flights` option. This menu definition is `EQUIPMNT.MNU`, a menu file called by `EQUIPMNT.WFM` on the `DBASEWIN\SAMPLES` directory:

```
PARAMETER FormObj
NEW EQUIPMNTMENU(FormObj,"Root")
CLASS EQUIPMNTMENU(FormObj,Name) OF MENU(FormObj,Name)
  this.Text = ""

  DEFINE MENU FILE OF THIS;
  PROPERTY;
  Text "&File"

  DEFINE MENU EXIT OF THIS.FILE;
  PROPERTY;
  OnClick {;form.close()},;
  Text "E&xit",;
  Shortcut "CTRL-Q"

  DEFINE MENU EQUIPMENT OF THIS;
  PROPERTY;
  Text "&Equipment"

  DEFINE MENU SELECT_FLIGHTS OF THIS.EQUIPMENT;
  PROPERTY;
  OnClick CLASS::GETFLIGHTS,,
  Text "&Select Flights",;
  Shortcut "CTRL-S"

PROCEDURE GETFLIGHTS
  LOCAL getFltsF, Selected
  SET PROCEDURE TO GetFlts.wfm ADDITIVE
  getFltsF = NEW GetFltsForm()
```

```

        getFltsF.MDI = .f.
        getFltsF.ReadModal()
        SHOW OBJECT form.FlightsBrowse
    RETURN
ENDCLASS

```

See Also

_app, CLASS MENUBAR, CLASS POPUP, DEFINE, MDI

CLASS MENUBAR

A MenuBar object specifies a top-level menu for a form. Using the MENUBAR class lets you add the standard Windows Edit and Windows pull-down menus to a form.

Properties

The following table lists the properties of the Menubar class. For more information on each property, see Chapter 8.

Property	Default	Description
ClassName	MENUBAR	Identifies the menubar object's class
EditCopyMenu	.F.	Specifies a menu item that copies selected text from a control to the Windows clipboard
EditCutMenu	.F.	Specifies a menu item that deletes selected text from a control and copies it to the Windows clipboard
EditPasteMenu	.F.	Specifies a menu item that pastes text from the Windows clipboard to the edit control with focus
EditUndoMenu	.F.	Specifies a menu item that restores the form to the state before the last edit operation was performed
ID	1	Identifies the menubar object with a numeric value
Name	MENUBAR1	Specifies the menubar object's name
OnInitMenu	N/A	Specifies code that executes when the menubar is accessed
Parent	N/A	An object reference that points to the parent form
Release()	N/A	Removes the MenuBar definition from memory
WindowMenu	.F.	Specifies a top-level menu that displays the Window List of all open MDI windows

Description

A Menubar object specifies a top-level menu for a form. A form's top-level menu doesn't contain any menu prompts itself; it is only the container for child menu objects. The child menu objects of the top-level menu contain the form's actual menu items.

Menu objects that have a Menubar as a Parent appear on the top line of a form. By default, the Menu Designer creates a MenuBar subclass when creating a .MNU file.

You can design and implement menus without using Menubars, as in earlier versions of dBASE. The advantage of using the MENUBAR class is that you can implement an Edit pulldown that uses the Windows clipboard for Cut, Copy, Paste and Undo operations,

and you can implement a Window pulldown that offers the standard MDI window list. The properties that enable these menu choices (EditCutMenu, EditCopyMenu, etc.) all take an object reference to a Menu object as their value.

To quickly add the Edit and Windows menus and their dropdown options (Cut, Copy, etc.), use the Menu Designer and add these options using the Menu pulldown menu.

Note The command CREATE MENU creates a Menubar subclass by. You can also use DEFINE MENUBAR m OF FormX to create a menu bar for the form named FormX.

Example

```

** END HEADER -- do not remove this line*
* Generated on 03/31/95
*
Parameter FormObj
NEW FOOMENU(FormObj, "Root")
CLASS FOOMENU(FormObj, Name) OF MENUBAR(FormObj, Name)
  DEFINE MENU FILE OF THIS;
    PROPERTY;
    Text "&File"
    DEFINE MENU EXIT OF THIS.FILE;
    PROPERTY;
    Text "E&xit"
  DEFINE MENU EDIT OF THIS;
    PROPERTY;
    Text "&Edit"
    DEFINE MENU UNDO OF THIS.EDIT;
    PROPERTY;
    Text "&Undo"
    DEFINE MENU CUT OF THIS.EDIT;
    PROPERTY;
    Text "Cu&t"
    DEFINE MENU COPY OF THIS.EDIT;
    PROPERTY;
    Text "&Copy"
    DEFINE MENU PASTE OF THIS.EDIT;
    PROPERTY;
    Text "&Paste"
  DEFINE MENU WINDOW OF THIS;
    PROPERTY;
    Text "&Window"
    DEFINE MENU ARRANGE OF THIS.WINDOW;
    PROPERTY;
    Text "&Arrange"
  DEFINE MENU HELP OF THIS;
    PROPERTY;
    Text "&Help"
    DEFINE MENU ABOUT OF THIS.HELP;
    PROPERTY;
    Text "&About"
  This.EditUndoMenu = This.Edit.Unundo
  This.EditCutMenu = This.Edit.Cut
  This.EditCopyMenu = This.Edit.Copy
  This.EditPasteMenu = This.Edit.Paste

```

```
        This.WindowMenu    = This.Window
    ENDCLASS
```

See Also
CLASS MENU, CLASS POPUP, DEFINE

CLASS OBJECT

Creates a custom object with no properties.

Properties
The Object class does not have built-in properties.

Description
Use the Object class to create your own object. An object of the Object class is empty—it contains no properties or methods. You customize this object by creating the properties you want.

Example
The following example creates a user-defined object that contains the name and address of a client:

```
oClient=NEW OBJECT()
oClient.Firstname = "Harvey"
oClient.Lastname = "West"
oClient.Address = "111 Last St."
oClient.City = "Portland"
oClient.State= "OR"
```

See Also
CLASS...ENDCLASS, CLASS PAINTBOX

CLASS OLE

Displays an OLE document that is stored in an OLE field, and lets the user initiate an action in the server application that created the document.

Properties
The following table lists the properties of the OLE class. For more information on each property, see Chapter 8.

Property	Default	Description
Before	N/A	Specifies which object the OLE object precedes in the tabbing order of the parent form
Border	.F.	Determines if the OLE object is surrounded with a border

Property	Default	Description
ClassName	OLE	Identifies the OLE class
DataLink	Empty string	Links the OLE object to a field
DoVerb()	N/A	Starts an OLE server session and determines its type
Enabled	.T.	Determines if the OLE object can be selected
Height	N/A	Sets the height
hWnd	N/A	Returns the OLE object handle
ID	-1	Identifies the OLE object with a numeric value
Left	N/A	Sets the position of the left border
LinkFileName	Empty string	Identifies which OLE document file (if any) is linked with the current OLE field.
Name	OLE1	Specifies the name of the OLE object
OleType	0	Returns a number that reveals whether an OLE field is empty, contains an embedded document, or contains a link to a document file.
OnChange	N/A	Executes a subroutine when the user modifies a document
OnClose	N/A	Executes a subroutine when the OLE server session is ended
OnGotFocus	N/A	Executes a subroutine when the OLE object receives focus
OnLostFocus	N/A	Executes a subroutine when focus is removed
OnOpen	N/A	Executes a subroutine when the parent form is opened
Parent	N/A	An object reference that points to the parent form
PageNo	N/A	Specifies on which page of a multi-page form the OLE object appears
Release()	N/A	Removes the OLE object definition from memory
ServerName	Empty string	Identifies the server application that is invoked when the user double-clicks on an OLE viewer object.
SetFocus()	N/A	Gives focus to the OLE object
StatusMessage	Empty string	Specifies a message to display on the status bar while the OLE object has focus
TabStop	.T.	Determines if the user can give object focus to the OLE object by pressing <i>Tab</i> or <i>Shift+Tab</i>
Top	N/A	Sets the position of the top border
Visible	.T.	Determines whether the OLE object is visible or hidden
Width	N/A	Sets the width

Description

Place an OLE object in a form to view and edit a document stored in an OLE field. For example, if an OLE field contains a bitmap image created in Paintbrush, double-clicking the OLE object linked to the field starts a session in Paintbrush and places the image in the Paintbrush work area.

OLE stands for Object Linking and Embedding. When you *link* a document to an OLE object, the OLE field does not contain the document itself; instead, it holds a link to a file containing the document. When you embed a document in an OLE field, a copy of the document is inserted into the OLE field, and no connection is made to a document file.

By double-clicking the OLE object, the user can invoke the application that created the OLE document. Therefore, if an image was created in Paintbrush and linked or embedded in the OLE field, double-clicking on the field starts a session in Paintbrush;

the image is displayed in the Paintbrush drawing area, ready for editing. If the object was linked, any changes made in the Paintbrush session are stored in the document file; if the object was embedded, the changes are stored in the OLE field only.

An OLE viewer window object displays the contents of an OLE field. (Use the `DataLink` property to identify this field by name.) Each time the record pointer is moved, the contents of the viewer window are refreshed to display the OLE field in the current record.

When you create an OLE object with the `NEW` operator, you can specify two parameters:

- *<parent form reference>*—An object reference pointing to the parent form.
- *<object name expC>*—A character string assigned to the `Name` property of the new OLE object. This value is optional.

For example, the following commands create a form and an OLE object to display in it:

```
MyForm = NEW FORM()
MyOLE = NEW OLE(MyForm, "OurOLE")
```

The `Name` property of the new OLE object contains "OurOLE".

Example

The following example is an extract from `PICTURES.WFM` on the `SAMPLES` directory and demonstrates displaying an OLE field from the `Pictures` table on a form with an OLE object:

```
LOCAL f
f = NEW PICTURES ()
f.Open()

CLASS PICTURES OF FORM
  this.EscExit = .T.
  this.View = "PICTURES.QBE"
  this.ColorNormal = "BG/B"
  this.Text = "Pictures Form"
  this.Width = 76.00
  this.Top = 0.00
  this.Left = 0.00
  this.Height = 30.00
  this.Minimize = .F.
  this.Maximize = .F.
  this.OnOpen = {;create session}

DEFINE PUSHBUTTON SOUND OF THIS;
PROPERTY;
  OnClick {;play sound binary pictures->sound},;
  Text "Sound",;
  Width 18.00,,
  Top 5.00,,
  Left 1.00,,
  Height 3.00,,
  FontSize 16.00,,
  FontName "Courier"
```

CLASS OLE

```
DEFINE LISTBOX THINGS OF THIS;
PROPERTY;
    ColorNormal "bg+/b",;
    Width       18.50,,;
    Top         11.42,,;
    Left        0.75,,;
    Height      5.50,,;
    DataSource  "FIELD NAME",;
    ColorHighLight "W+/B",;
    FontSize    11.25,,;
    FontName    "Fixedsys",;
    ID          800

DEFINE OLE PICTURE OF THIS;
PROPERTY;
    Width       55.00,,;
    Top         5.00,,;
    Left        20.00,,;
    Height      24.00,,;
    DataLink    "PICTURES->BITMAPOLE",;
    ID          88

DEFINE TEXT TITLE OF THIS;
PROPERTY;
    ColorNormal "gr+/b",;
    Text "Sights and Sounds",;
    Width       59.50,,;
    Top         0.00,,;
    Left        20.00,,;
    Height      4.30,,;
    FontSize    32.00,,;
    FontName    "Serif"
ENDCLASS

PROCEDURE Sound_OnClick
PLAY SOUND BINARY Pictures->Sound

PROCEDURE ClosePictures
USE IN PICTURES
FORM.CLOSE()
```

See Also

CLASS AUTOCLIENT, CLASS DDELINK, CLASS DDETOPIC, CLASS IMAGE,
DEFINE, DoVerb

CLASS OLEAUTOCLIENT

Creates an OLE2 controller which attaches to an OLE2 server.

Properties

The properties of this class are determined by the server.

Description

Use CLASS OLEAUTOCLIENT to attach to a server program. The syntax is:

```
<ClientClassName> = NEW OLEAUTOCLIENT<exp>
```

where <exp> is the server program ID. There is no equivalent DEFINE OLEAUTOCLIENT statement.

After you have created the class, you can use the Property Inspector to see its properties. You can change properties by using the Inspector or by using standard ClientClassName.property statements.

Example

```
*
*  OLEWORD.PRG
*  Sample program to illustrate OLE2 Automation
*  with Microsoft Word as the server.
*
*
*  Create OLE2 Automation object. The parameter
*  is the ProgID
*
ww = new oleautoclient("word.basic")
*
*  All properties and methods of the OLE2
*  Automation object are documented by the
*  server.
*
ww.FileNew("Normal", 0)
ww.Insert("This is my configuration file")
ww.InsertBreak(6)
ww.InsertBreak(6)
ww.InsertFile("c:\config.sys")
ww.StartOfDocument()
ww.EndOfLine(1)
ww.EditCut()
ww.EditPaste()
ww.EditPaste()
? "current font size is", ww.FontSize()
ww.EditSelectAll()
ww.GrowFont()
ww.GrowFont()
ww.GrowFont()
```

```
? "font size is now ", ww.FontSize()
ww.EditCopy()      && Can paste into dBASE later
ww.FilePrint()
*
* Uncomment the following line to
* close Word
*ww.AppClose()
```

See Also
CLASS DDELINK, CLASS DDETOPIC, CLASS OLE

CLASS PAINTBOX

A generic control that can be placed on a form.

Properties

The following table lists the properties of the Paintbox class. For more information on each property, see Chapter 8.

Property	Default	Description
Before	N/A	Specifies which object the paintbox object precedes in the tabbing order of the parent form
ClassName	PAINTBOX	Identifies the paintbox object's class
ColorNormal	WindowText/ Window	Sets the color of the paintbox object when it isn't highlighted
Enabled	.T.	Determines if the paintbox object can be selected
Height	N/A	Sets the height
HelpFile	Empty string	Identifies a Windows Help file (.HLP) that contains context-sensitive Help topics
HelpID	Empty string	Specifies the context string or context number of a Help topic in a Windows Help file (.HLP)
hWnd	N/A	Returns the paintbox object handle
ID	-1	Identifies the paintbox object with a numeric value
Left	N/A	Sets the position of the left border
Move()	N/A	Moves or sizes the paintbox object
Name	PAINTBOX1	Specifies the paintbox object's name
OnChar	N/A	Executes a subroutine when a "printable" key or key combination is pressed
OnFormSize	N/A	Executes a subroutine whenever the parent form is resized, restored, or maximized
OnGotFocus	N/A	Executes a subroutine when the paintbox object receives focus
OnKeyDown	N/A	Executes a subroutine when any key is pressed
OnKeyUp	N/A	Executes a subroutine when any key is released
OnLeftDblClick	N/A	Executes a subroutine when the user double-clicks the paintbox object

Property	Default	Description
OnLeftMouseDown	N/A	Executes a subroutine when the user clicks the paintbox object with the left mouse button
OnLeftMouseUp	N/A	Executes a subroutine when the user releases the left mouse button while the pointer is over the paintbox object
OnLostFocus	N/A	Executes a subroutine when focus is removed
OnMiddleDbClick	N/A	Executes a subroutine when the user double-clicks the paintbox object with the middle mouse button
OnMiddleMouseDown	N/A	Executes a subroutine when the user clicks the paintbox object with the middle mouse button
OnMiddleMouseUp	N/A	Executes a subroutine when the user releases the middle mouse button while the pointer is on the paintbox object
OnMouseMove	N/A	Executes a subroutine when the user moves the mouse over the paintbox object
OnOpen	N/A	Executes a subroutine when the parent form is opened
OnPaint	N/A	Executes a subroutine whenever the object needs to be redrawn
OnRightDbClick	N/A	Executes a subroutine when the user double-clicks the paintbox object with the right mouse button
OnRightMouseDown	N/A	Executes a subroutine when the user clicks the paintbox object with the right mouse button
OnRightMouseUp	N/A	Executes a subroutine when the user releases the right mouse button while the pointer is on the paintbox object
PageNo	N/A	Specifies on which page of a multi-page form the paintbox object appears
Parent	N/A	An object reference that points to the parent form
Release()	N/A	Removes the paintbox object definition from memory
SetFocus()	N/A	Gives focus to the paintbox object
TabStop	.T.	Determines if the user can give object focus to the paintbox object by pressing Tab or Shift+Tab
Top	N/A	Sets the position of the top border
Visible	.T.	Determines whether the paintbox object is visible or hidden
Width	N/A	Sets the width

Description

The PaintBox object is a generic control you can use to create a variety of objects. It is designed for advanced developers who want to create their own custom controls using the Windows API. It is simply a rectangular region of a form which has all the standard control properties such as Height, Width, and Before, as well as all the standard mouse events.

In addition to the standard events or properties, the PaintBox object has three events that let you detect keystrokes entered when it has focus: OnChar, OnKeyDown, and OnKeyUp. These let you create customized editing controls. The OnPaint and OnFormSize properties let you modify the appearance of the object based on user interaction.

Example

```

local f
f = new PAINTEXFORM()
f.Open()
CLASS PAINTEXFORM OF FORM
  this.OnLeftMouseUp = CLASS::FORM_ONLEFTMOUSEUP
  this.Text = "Form"
  this.Left = 54.5
  this.Top = 2
  this.PageNo = 1
  this.ColorNormal = "N/BTNFACE"
  this.Height = 20.6465
  this.TopMost = .F.
  this.Width = 67.666
DEFINE PAINTBOX PAINTBOX1 OF THIS;
  PROPERTY;
    OnPaint CLASS::PAINTBOX1_ONPAINT;;
    OnLeftMouseDown CLASS::PAINTBOX1_ONLEFTMOUSEDOWN;;
    OnLeftMouseUp CLASS::PAINTBOX1_ONLEFTMOUSEUP;;
    Left 9.333;;
    Top 2;;
    ColorNormal "B+/0xffff80",;
    PageNo 1;;
    Height 6.8818;;
    OnOpen CLASS::PAINTBOX1_ONOPEN;;
    Width 21.333
  Procedure PAINTBOX1_OnPaint
    hfact = (256/43)
    vfact = (256/15.5)
    lwidth = form.paintbox1.width * hfact
    lheight = form.paintbox1.height * vfact
    form.pointarray = makepoint(lwidth/2,0)
    form.pointarray = form.pointarray + makepoint(0,lheight)
    form.pointarray = form.pointarray + makepoint(lwidth,lheight)
    local hDC
    hDC = GetDc(this.hwnd)
    hBrush = CreateSolidBrush(hdc,RGB(255,0,0))
    SelectObject(hdc,hbrush)
    SetTextColor(hdc, RGB(0,0,255))
    SetPolyFillMode(hdc,2)
    Polygon(hdc,form.pointarray,3)
    ReleaseDc(this.hWnd, hdc)
  return
  Procedure PAINTBOX1_OnOpen
    set proc to program(1) ADDITIVE

    EXTERN CHANDLE  GetDc(CHANDLE) USER.EXE
    EXTERN CINT     ReleaseDc(CHANDLE,CHANDLE) USER.EXE
    EXTERN CLOGICAL Polygon(CHANDLE,CPTR,CINT) GDI.EXE
    EXTERN CINT     SetTextColor(CHANDLE, CLONG) GDI.EXE
    EXTERN CLOGICAL Ellipse(CHANDLE,CINT,CINT,CINT,CINT) GDI.EXE
    EXTERN CLOGICAL FloodFill(chandle,cint,cint,clong) GDI.EXE
    EXTERN CINT     SetPolyFillMode(chandle,cint) GDI.EXE

```

```
        EXTERN Chandle  CreateSolidBrush(chandle,clong) GDI.EXE
        EXTERN Chandle  SelectObject(chandle,chandle) GDI.EXE
        this.moving = .f.
    return

    Procedure PAINTBOX1_OnLeftMouseDown(flags, col, row)
        this.moving = .t.
    return

    Procedure form_OnLeftMouseUp(flags, col, row)
        if form.paintbox1.moving

form.paintbox1.move(col,row,form.paintbox1.width,form.paintbox1.height
)
        form.paintbox1.moving = .f.
    endif
    return

    Procedure PAINTBOX1_OnLeftMouseUp(flags, col, row)
        this.moving = .f.
    return
ENDCLASS
function RGB(r, g, b)
return b*65536+g*256+r
function MakePoint(x,y)
return(MakeInt(x) + MakeInt(y))
function MakeRect(left, top, width, height)
return
MakeInt(left)+MakeInt(top)+MakeInt(width+left)+MakeInt(top+height)
function MakeInt(int)
return chr(bitand(int,255))+chr(bitr(int,8))
```

See Also
CLASS IMAGE, CLASS OBJECT

CLASS POPUP

A Windows-style popup menu assigned to a form.

Properties

The following table lists the properties of the Popup class. For more information on each property, see Chapter 8.

Property	Default	Description
ClassName	POPUP	Identifies the Popup class
ID	1	Identifies the popup with a numeric value
Left	N/A	Sets the position of the left border
Name	POPUP1	Specifies the name of the popup menu
OnInitMenu	N/A	Specifies code that executes when the popup menu is opened
Open()	N/A	Opens the popup menu

Property	Default	Description
Parent	N/A	An object reference that points to the parent form
Release()	N/A	Removes the popup definition from memory
Top	N/A	Sets the position of the top border
TrackRight	.T.	Determines whether the popup menu responds to a right mouse click for selection of a menu item

Description

Use CLASS POPUP to add a popup menu to a form. Popup menus give users a "shortcut" way to perform actions without pulling down items from the menu bar. A Popup's parent is always a Form. Individual menu items are attached to a Popup by defining Menu objects with the Popup as parent.

A .POP file is similar to a .MNU in format with the name of the Popup passed as a parameter instead of the explicit "Root" used in .MNU files.

A popup menu consists of two elements:

- The object reference variable that identifies the entire popup menu. You must create this variable before you can create the menu. (The variable name is not displayed anywhere.)
- Menu items, the prompts offered by the popup menu.

When you create a popup menu with the NEW operator, you can specify two parameters:

- *<parent form reference>*-An object reference pointing to the parent form.
- *<object name expC>*-A character string assigned to the Name property of the new popup menu. This value is optional.

For example, the following commands create a form and a popup to display in it:

```
MyForm = NEW FORM()
MyForm.MyPop = NEW POPUP(MyForm, "OurPopup")
MyForm.MyPop.Item1 = NEW MENU(MyForm.MyPop, "Close")
MyForm.MyPop.Item2 = NEW MENU(MyForm.MyPop, "Close and Save")
MyForm.Open()
MyForm.MyPop.Open()
```

The Name property of the new popup object contains "OurPopup".

Assigning actions to popup menu items You assign an action to a popup menu item with the OnClick subroutine. For example, the following command assigns a subroutine named ClsSave to the Close and Save menu item of the example above:

```
MyForm.MyPop.Item2.OnClick = ClsSave
```

Note You can design a popup menu with the Popup Designer, a tool that a popup menu file (.POP). A popup menu file contains dBASE code that generates the popup menu you design. To access the Popup Designer, click the File menu and select New | Popup.

Example

```
f = NEW Form()
DEFINE POPUP p OF f
DEFINE MENU Inspect OF f.p;
| PROPERTY;
|   Text "Inspector"
```

See Also

CLASS MENU

CLASS PUSHBUTTON

A button that executes a command or action when a user chooses it.

Properties

The following table lists the properties of the Pushbutton class. For more information on each property, see Chapter 8.

Property	Default	Description
Before	N/A	Specifies which object the pushbutton precedes in the tabbing order of the parent form
ClassName	PUSHBUTTON	Identifies the Pushbutton class
ColorNormal	BtnText/BtnFace	Sets the color of the pushbutton
Default	.F.	Determines if the pushbutton is the default pushbutton
DisabledBitmap	Empty string	Specifies the graphic image to display in the pushbutton when the pushbutton is disabled
DownBitmap	Empty string	Specifies the graphic image to display in the pushbutton when the user presses the mouse button over the pushbutton, or when the user presses the <i>Spacebar</i> while the pushbutton has focus
Enabled	.T.	Determines if the pushbutton can be selected
FocusBitmap	Empty string	Specifies the graphic image to display in the pushbutton when the pushbutton has focus
FontBold	.T.	Determines if characters are displayed in bold type
FontItalic	.F.	Determines if characters are displayed in italic type
FontName	MS Sans Serif	Specifies the font to apply to displayed characters
FontSize	N/A	Specifies the size of the font in points
FontStrikeOut	.F.	Determines if characters are displayed in strikeout type
FontUnderline	.F.	Determines if characters are displayed in underlined type
Group	.T.	Starts an object group in the parent form
Height	N/A	Sets the height
HelpFile	Empty string	Identifies a Windows Help file (.HLP) that contains context-sensitive Help topics
HelpID	Empty string	Specifies the context string or context number of a Help topic in a Windows Help file (.HLP)
hWnd	N/A	Returns the object handle of the pushbutton

Property	Default	Description
ID	-1	Identifies the pushbutton with a numeric value
Left	N/A	Sets the position of the left border
MousePointer	0	Specifies the mouse pointer type when the pointer is over the pushbutton
Move()	N/A	Moves or sizes the pushbutton
Name	PUSHBUTTON1	Specifies the name of the pushbutton
OnClick	N/A	Executes a subroutine when the user chooses a pushbutton
OnGotFocus	N/A	Executes a subroutine when the pushbutton receives focus
OnHelp	N/A	Executes a subroutine when the user presses <i>F1</i>
OnLeftDblClick	N/A	Executes a subroutine when the user double-clicks the pushbutton
OnLeftMouseDown	N/A	Executes a subroutine when the user clicks the pushbutton with the left mouse button
OnLeftMouseUp	N/A	Executes a subroutine when the user releases the left mouse button while the pointer is over the pushbutton
OnLostFocus	N/A	Executes a subroutine when focus is removed
OnMiddleDblClick	N/A	Executes a subroutine when the user double-clicks the pushbutton with the middle mouse button
OnMiddleMouseDown	N/A	Executes a subroutine when the user clicks the pushbutton with the middle mouse button
OnMiddleMouseUp	N/A	Executes a subroutine when the user releases the middle mouse button while the pointer is on the pushbutton
OnMouseMove	N/A	Executes a subroutine when the user moves the mouse in the pushbutton
OnOpen	N/A	Executes a subroutine when the parent form is opened
OnRightDblClick	N/A	Executes a subroutine when the user double-clicks the pushbutton with the right mouse button
OnRightMouseDown	N/A	Executes a subroutine when the user clicks the pushbutton with the right mouse button
OnRightMouseUp	N/A	Executes a subroutine when the user releases the right mouse button while the pointer is on the pushbutton
PageNo	N/A	Specifies on which page of a multi-page form the pushbutton object appears
Parent	N/A	An object reference that points to the parent form
Release()	N/A	Removes the pushbutton definition from memory
SetFocus()	N/A	Gives focus to the pushbutton
SpeedBar	.F.	Determines whether the pushbutton behaves like a SpeedBar button or a standard pushbutton
SpeedTip	Empty string	Specifies the text that appears when the mouse remains on the pushbutton for more than one second
StatusMessage	Empty string	Specifies a message to display on the status bar while the pushbutton has focus
TabStop	.T.	Determines if the user can give object focus to the pushbutton by pressing <i>Tab</i> or <i>Shift+Tab</i>
Text	N/A	Specifies a character string to display on the pushbutton
Top	N/A	Sets the position of the top border
UpBitmap	Empty string	Specifies the graphic image to display in the pushbutton when it isn't selected

Property	Default	Description
Visible	.T.	Determines whether the pushbutton is visible or hidden
When	N/A	Specifies a condition that must evaluate to true before the user can give focus to the pushbutton
Width	N/A	Sets the width

Description

Use a pushbutton to execute a specific action when the user chooses it.

When the user chooses a pushbutton, the following things happen:

- dBASE executes any procedure or codeblock you assign to the OnGotFocus property.
- dBASE executes any procedure or codeblock you assign to the OnClick property.
- dBASE *submits* the parent form, which executes any procedure or codeblock you assign to the OnSelection property of the parent form.

The OnSelection procedure or codeblock identifies the pushbutton through the ID property. When the user chooses the pushbutton, dBASE passes this value to the procedure or codeblock you assigned to the OnSelection property. The procedure or codeblock can use Id to identify which object was last selected.

When you create a pushbutton with the NEW operator, you can specify two parameters:

- *<parent form reference>*—An object reference pointing to the parent form.
- *<object name expC>*—A character string assigned to the Name property of the new pushbutton. This value is optional.

For example, the following commands create a form and a pushbutton to display in it:

```
MyForm = NEW FORM()
MyButton = NEW PUSHBUTTON(MyForm, "OurButton")
```

The Name property of the new pushbutton contains "OurButton".

Note You can write an OnClick or OnSelection procedure with the Procedure Editor, a window that lets you enter dBASE program code. To access the Procedure Editor, click the Tool Button next to the OnClick or OnSelection item in the Inspector.

Example

The following example defines a form with two fields from the Contact table displayed and three alternative methods of defining pushbuttons to advance or retard the record pointer and exit the form:

```
SET PROCEDURE TO BUTTONS.CC ADDITIVE
* Makes available custom objects PREVBUTTON,
* NEXTBUTTON and CANCELBUTTON
LOCAL f
f=NEW EntryForm()
f.OPEN()
CLASS EntryForm OF FORM
  this.Top=2
  this.Left=2
```

CLASS PUSHBUTTON

```
this.Width=36
this.Height=27
this.View = "Contact.DBF"
this.Text= "Class Pushbutton Demo"
DEFINE ENTRYFIELD CompCode OF THIS;
    PROPERTY;
        Width 5;;
        Top 1;;
        Left 2;;
        Height 1.5;;
        FontBold .T.;;
        DataLink "COMPCODE"
DEFINE ENTRYFIELD Contact OF THIS;
    PROPERTY;
        Width 22;;
        Top 1;;
        Left 12;;
        Height 1.5;;
        FontBold .T.;;
        DataLink "CONTACT"
DEFINE PUSHBUTTON Reverse OF THIS;
    PROPERTY;
        OnClick {;SKIP-1},;
        Text "&Previous",;
        Top 4;;
        Left 5;;
        Width 12;;
        Height 2.5;;
        Group .T.;;
        FontSize 8;;
        FocusBitmap "Resource #104 DBAS0009.DLL",;
        DownBitmap "Resource #104 DBAS0009.DLL",;
        UpBitmap "Resource #104 DBAS0009.DLL"
DEFINE PUSHBUTTON NxtRcd OF THIS;
    PROPERTY;
        OnClick {;SKIP},;
        Text "&Next",;
        Top 4;;
        Left 19;;
        Width 12;;
        Height 2.5;;
        Group .T.;;
        FontSize 8;;
        FocusBitmap "Resource #100 DBAS0009.DLL",;
        DownBitmap "Resource #100 DBAS0009.DLL",;
        UpBitmap "Resource #100 DBAS0009.DLL"
DEFINE PUSHBUTTON Cx OF THIS;
    PROPERTY;
        OnClick {;Form.Close()},;
        Text "&Cancel",;
        Top 7;;
        Left 12;;
        Height 2.5;;
        Width 12;;
        Group .T.;;
```

```

        FontSize 8;;
        FocusBitmap "Resource #28 DBAS0009.DLL",;
        DownBitmap "Resource #29 DBAS0009.DLL",;
        UpBitmap "Resource #28 DBAS0009.DLL"
    DEFINE LINE Ln1 OF THIS;
        PROPERTY Left 7, Top 11, Width 4,;
            Bottom 11, Right 29, ColorNormal "RB/W"
    DEFINE PUSHBUTTON Back OF THIS AT 13,7;
        PROPERTY Text "&Back", Height 2,;
            OnClick {;SKIP-1}, FontBold .T.
    DEFINE PUSHBUTTON Next OF THIS AT 13,21;
        PROPERTY TEXT "&Next", Height 2,;
            OnClick {;SKIP}, FontBold .T.
    DEFINE PUSHBUTTON Exit OF THIS AT 16,14;
        PROPERTY Text "&Exit", Height 2,;
            OnClick {;Form.Close()}, FontBold .T.
    DEFINE LINE Ln2 OF THIS;
        PROPERTY Left 7, Top 19, Width 4,;
            Bottom 19, Right 29, ColorNormal "RB/W"
    DEFINE PREVBUTTON PriorRecord OF THIS;
        PROPERTY;
            Top 21 ,;
            Left 6
    DEFINE NEXTBUTTON NextRecord OF THIS;
        PROPERTY;
            Top 21,;
            Left 20
    DEFINE CANCELBUTTON Cx OF THIS;
        PROPERTY;
            Top 24,;
            Left 13
ENDCLASS

```

See Also
 CLASS FORM, DEFINE, ON SELECTION FORM

CLASS RADIOBUTTON

An object that represents a single choice in a set of mutually exclusive choices.

Properties
 The following table lists the properties of the Radiobutton class. For more information on each property, see Chapter 8.

Property	Default	Description
Before	N/A	Specifies which object the radio button precedes in the tabbing order of the parent form
ClassName	RADIOBUTTON	Identifies the radio button class
ColorNormal	BtnText/BtnFace	Sets the color of the radio button
DataLink	Empty string	Links the radio button to a field

Property	Default	Description
Enabled	.T.	Determines if the radio button can be selected
FontBold	.T.	Determines if characters in the radio button prompt are displayed in bold type
FontItalic	.F.	Determines if characters are displayed in italic type
FontName	MS Sans Serif	Specifies the font to apply to displayed characters
FontSize	N/A	Specifies the size of the font in point size
FontStrikeOut	.F.	Determines if characters are displayed in strikeout type
FontUnderline	.F.	Determines if characters are displayed in underlined type
Group	N/A	Starts an object group in the parent form
Height	N/A	Sets the height
HelpFile	Empty string	Identifies a Windows Help file (.HLP) that contains context-sensitive help topics
HelpID	Empty string	Specifies the context string or context number of a help topic in a Windows Help file (.HLP)
hWnd	N/A	Returns the radio button handle
ID	-1	Identifies the radio button with a numeric value
Left	N/A	Sets the position of the left border
MousePointer	0	Specifies the mouse pointer type when the pointer is over the radio button
Move()	N/A	Moves or sizes the radio button
Name	RADIOBUTTON1	Specifies the name of the radio button
OldStyle	.F.	Determines if the radio button is displayed in the default Windows style or in dBase style
OnChange	N/A	Executes a subroutine when the user selects a different radio button
OnGotFocus	N/A	Executes a subroutine when the radio button receives focus
OnHelp	N/A	Executes a subroutine when the user presses <i>F1</i>
OnLeftDblClick	N/A	Executes a subroutine when the user double-clicks the radio button
OnLeftMouseDown	N/A	Executes a subroutine when the user clicks the radio button with the left mouse button
OnLeftMouseUp	N/A	Executes a subroutine when the user releases the left mouse button while the pointer is over the radio button
OnLostFocus	N/A	Executes a subroutine when focus is removed
OnMiddleDblClick	N/A	Executes a subroutine when the user double-clicks the radio button with the middle mouse button
OnMiddleMouseDown	N/A	Executes a subroutine when the user clicks the radio button with the middle mouse button
OnMiddleMouseUp	N/A	Executes a subroutine when the user releases the middle mouse button while the pointer is on the radio button
OnMouseMove	N/A	Executes a subroutine when the user moves the mouse in the radio button
OnOpen	N/A	Executes a subroutine when the parent form is opened
OnRightDblClick	N/A	Executes a subroutine when the user double-clicks the radio button with the right mouse button
OnRightMouseDown	N/A	Executes a subroutine when the user clicks the radio button with the right mouse button

Property	Default	Description
OnRightMouseUp	N/A	Executes a subroutine when the user releases the right mouse button while the pointer is on the radio button
PageNo	N/A	Specifies on which page of a multi-page form the radio button object appears
Parent	N/A	An object reference that points to the parent form
Release()	N/A	Removes the radio button definition from memory
SetFocus()	N/A	Gives focus to the radio button
SpeedTip	Empty string	Specifies the text that appears when the mouse remains on the radio button for more than one second
StatusMessage	Empty string	Specifies a message to display on the status bar while the radio button has focus
TabStop	.T.	Determines if the user can give focus to the radio button by pressing <i>Tab</i> or <i>Shift+Tab</i>
Text	N/A	Specifies a character string to display next to the radio button
Top	N/A	Sets the position of the top border
Value	N/A	Determines whether the radio button is checked
Visible	.T.	Determines whether the radio button is visible or hidden
When	N/A	Specifies a condition that must evaluate to true before the user can give focus to the radio button
Width	N/A	Sets the width

Description

Use a radio button group to make users choose between options you provide, like the buttons on older car radios. The user can select only one option from a group; for example, a report-printing application might use a three-button group that gives a choice between "Printer", "Screen", or "File". For radio buttons to work properly, they must be arranged in such groups.

Use the Group property to make two or more radio buttons work together. For example, if you create seven radio buttons and set the Group properties of the first and fourth radio button to true (.T.), the first three buttons form one group, and the last four form another. The two groups are independent; the user can select one button in the first group and one button in the other.

Use the DataLink property to link the radio button with a field. Use the Text property to specify the value each radio button inserts into the field when chosen. The Text value is automatically displayed as a prompt next to the radio button. The text property contains character data, so you can set DataLink to character fields only.

The Value property, which contains a logical value of true (.T.) or false (.F.), indicates which button is selected from a group. The Value property of the selected radio button is true, while the Value properties of unselected buttons in the group are false. Use a CASE...ENDCASE statement to detect which radiobutton in the group is chosen.

When you create a radio button with the NEW operator, you can specify two parameters:

- *<parent form reference>*—An object reference pointing to the parent form.

- *<object name expC>*—A character string assigned to the Name property of the new radio button. This value is optional.

For example, the following commands create a form and a radio button to display in it:

```
MyForm = NEW FORM()
MyRadio = NEW RADIOBUTTON(MyForm, "OurRadio")
```

The Name property of the new radio button contains "OurRadio".

Note You can specify a field for the DataLink property with the Choose Field dialog box. To access this dialog box, click on the Tool Button next to the DataLink item in the Inspector.

Example

The following example creates an entry form with three radio buttons to select a desired conversion factor. Numeric values entered in the entry field are converted to the checked unit of measure after clicking the Compute pushbutton:

```
** Metric Conversion Program **
SET PROCEDURE TO PROGRAM(1) ADDITIVE
LOCAL f
f=NEW Convert()
f.OPEN()
CLASS Convert OF FORM
  this.Top=2
  this.Left=2
  this.Width=38
  this.Height=18
  this.Text= "Conversion Utility"
  DEFINE ENTRYFIELD Amt OF THIS;
    PROPERTY Value 0, Width 8;;
    Top 4, Left 15
  DEFINE TEXT Ln1 OF THIS;
    PROPERTY;
    Text "Enter Amount; Select a RadioButton",;
    Width 40, Top 2, Left 6
  DEFINE RadioButton Inches OF THIS;
    PROPERTY Text "Inches to Centimeters",;
    Width 22, Value .F., Top 6, Left 8
  DEFINE RadioButton Pounds OF THIS;
    PROPERTY Text "Pounds to Kilograms",;
    Width 21, Value .F., Top 8, Left 8
  DEFINE RadioButton Degrees OF THIS;
    PROPERTY Text "Degrees F to C",;
    Width 21, Value .F., Top 10, Left 8
  DEFINE TEXT Ln2 OF THIS;
    PROPERTY Text "Results:",;
    Width 30, Top 12, Left 8;;
    ColorNormal "R/W"
  DEFINE PUSHBUTTON Results OF THIS;
    PROPERTY TEXT "&Compute",;
    Top 15, Left 15;;
    OnClick {;myResult=Metric(form);
    ;Form.Ln2.Text="Results: " + myResult}
```



```

ENDCLASS

FUNCTION Metric(pForm)
DO CASE
CASE pForm.Inches.Value
myResult = LTRIM(STR(pForm.Amt.Value;
* 2.54,10,2))+ " Centimeters"
CASE pForm.Pounds.Value
myResult = LTRIM(STR(pForm.Amt.Value;
* .454,10,2))+ " Kilograms"
CASE pForm.Degrees.Value
myResult = LTRIM(STR((pForm.Amt.Value;
-32)* (5/9),10,2))+ " Degrees C"
ENDCASE
RETURN myResult

```

See Also
DEFINE

CLASS RECTANGLE

A rectangle you display at a specified location in a form.

Properties

The following table lists the properties of the Rectangle class. For more information on each property, see Chapter 8.

Property	Default	Description
Before	N/A	Specifies which object the rectangle object precedes in the tabbing order of the parent form
Border	.T.	Determines if the rectangle object is surrounded with a border
BorderStyle	0 (Normal)	Determines whether the border is Normal, Raised, or Lowered
ClassName	RECTANGLE	Identifies the rectangle class
ColorNormal	BtnText/ BtnFace	Sets the color of the rectangle object
FontBold	.T.	Determines if characters in the label of the rectangle object are displayed in bold type
FontItalic	.F.	Determines if characters in the label of the rectangle object are displayed in italic type
FontName	MS Sans Serif	Specifies the font to apply to displayed characters
FontSize	N/A	Specifies the size of the font in points
FontStrikeOut	.F.	Determines if characters in the label of the rectangle object are displayed in strikeout type
FontUnderline	.F.	Determines if characters in the label of the rectangle object are displayed in underlined type
Height	N/A	Sets the height
hWnd	N/A	Returns the object handle of the rectangle object

Property	Default	Description
Left	N/A	Sets the position of the left border
MousePointer	0	Specifies the mouse pointer type when the pointer is over the rectangle object
Move()	N/A	Moves or sizes the rectangle object
Name	RECTANGLE1	Specifies the name of the rectangle object
OldStyle	.F.	Determines if the rectangle object is displayed in the default Windows style or in dBase style
OnLeftDbClick	N/A	Executes a subroutine when the user double-clicks the rectangle object
OnLeftMouseDown	N/A	Executes a subroutine when the user clicks the rectangle object with the left mouse button
OnLeftMouseUp	N/A	Executes a subroutine when the user releases the left mouse button while the pointer is over the rectangle object
OnMiddleDbClick	N/A	Executes a subroutine when the user double-clicks the rectangle object with the middle mouse button
OnMiddleMouseDown	N/A	Executes a subroutine when the user clicks the rectangle object with the middle mouse button
OnMiddleMouseUp	N/A	Executes a subroutine when the user releases the middle mouse button while the pointer is on the rectangle object
OnMouseMove	N/A	Executes a subroutine when the user moves the mouse in the rectangle object
OnOpen	N/A	Executes a subroutine when the parent form is opened
OnRightDbClick	N/A	Executes a subroutine when the user double-clicks the rectangle object with the right mouse button
OnRightMouseDown	N/A	Executes a subroutine when the user clicks the rectangle object with the right mouse button
OnRightMouseUp	N/A	Executes a subroutine when the user releases the right mouse button while the pointer is on the rectangle object
PageNo	N/A	Specifies on which page of a multi-page form the rectangle object appears
Parent	N/A	An object reference that points to the parent form
PatternStyle	0 (Solid)	Specifies a pre-defined Windows background hatching pattern
Release()	N/A	Removes the rectangle object definition from memory
Text	N/A	Specifies a character string to display in the label of the rectangle object
Top	N/A	Sets the position of the top border
Visible	.T.	Determines whether the rectangle object is visible or hidden
Width	N/A	Sets the width

Description

Use a rectangle object to enclose an area of a form. For example, you can use a rectangle object to draw a border around a group of related objects, such as a group of radio buttons.

To assign a label that describes the group of objects, use the Text property. The label appears in the top left corner of the rectangle.

A rectangle object does not affect other objects. The user can't give focus to the rectangle object, and it doesn't display or modify data.

When you create a rectangle object with the NEW operator, you can specify two parameters:

- *<parent form reference>*—An object reference pointing to the parent form.
- *<object name expC>*—A character string assigned to the Name property of the new rectangle object. This value is optional.

For example, the following commands create a form and a rectangle object to display in it:

```
MyForm = NEW FORM()
MySquare = NEW RECTANGLE(MyForm, "OurSquare")
```

The Name property of the new rectangle object contains "OurSquare".

Example

The following example uses DEFINE RECTANGLE, within a Class definition, to create a lowered-border rectangle in the center of the form, which creates the visual effect of a raised border around the edge of the form:

```
LOCAL f
f=NEW DISPLAY()
f.OPEN()
CLASS DISPLAY OF FORM
  this.Top=2
  this.Left=2
  this.Width=36
  this.Height=13
  this.Text= "Class Rectangle Demo"
  DEFINE RECTANGLE Rec1 OF THIS;
    PROPERTY;
    Left 3;;
    Top 2;;
    Width 30;;
    Height 9;;
    ColorNormal "RB/W";
    BorderStyle 2      && Lowered border style
  DEFINE TEXT Txt1 OF THIS;
    PROPERTY;
    Text "Visual dBASE";
    Top 4, Left 5;;
    FontItalic .T., FontSize 14;;
    Width 19, Height 3, FontBold .T.;;
    ColorNormal "RB/W"
  DEFINE TEXT Txt2 OF THIS;
    PROPERTY;
    Text "has arrived";
    Top 7, Left 10;;
    FontItalic .T., FontSize 14;;
    Width 18, Height 3, FontBold .T.
ENDCLASS
```

See Also
CLASS LINE, CLASS SHAPE

CLASS SCROLLBAR

An object that lets the user increase or decrease a value by moving a slider button.

Properties

The following table lists the properties of the Scrollbar class. For more information on each property, see Chapter 8.

Property	Default	Description
Before	N/A	Specifies which object the scroll bar precedes in the tabbing order of the parent form
ClassName	SCROLLBAR	Identifies the scrollbar class
ColorNormal	BtnText/ BtnFace	Sets the color of the scroll bar
DataLink	Empty string	Links the scroll bar to a field
Enabled	.T.	Determines if the scroll bar can be selected
Height	N/A	Sets the height
HelpFile	Empty string	Identifies a Windows Help file (.HLP) that contains context-sensitive help topics
HelpID	Empty string	Specifies the context string or context number of a help topic in a Windows Help file (.HLP)
hWnd	N/A	Returns the scroll bar object handle
ID	-1	Identifies the scroll bar with a numeric value
Left	N/A	Sets the position of the left border
MousePointer	0	Specifies the mouse pointer type when the pointer is over the scroll bar
Move()	N/A	Moves or sizes the scroll bar
Name	SCROLLBAR1	Specifies the name of the scrollbar
OnChange	N/A	Executes a subroutine when the user moves the slider button
OnGotFocus	N/A	Executes a subroutine when the scroll bar receives focus
OnHelp	N/A	Executes a subroutine when the user presses <i>F1</i>
OnLeftDbtClick	N/A	Executes a subroutine when the user double-clicks the scroll bar
OnLeftMouseDown	N/A	Executes a subroutine when the user clicks the scroll bar with the left mouse button
OnLeftMouseUp	N/A	Executes a subroutine when the user releases the left mouse button while the pointer is over the scroll bar
OnLostFocus	N/A	Executes a subroutine when focus is removed
OnMiddleDbtClick	N/A	Executes a subroutine when the user double-clicks the scroll bar with the middle mouse button
OnMiddleMouseDown	N/A	Executes a subroutine when the user clicks the scroll bar with the middle mouse button
OnMiddleMouseUp	N/A	Executes a subroutine when the user releases the middle mouse button while the pointer is on the scroll bar

Property	Default	Description
OnMouseMove	N/A	Executes a subroutine when the user moves the mouse in the scroll bar
OnOpen	N/A	Executes a subroutine when the parent form is opened
OnRightDbClick	N/A	Executes a subroutine when the user double-clicks the scroll bar with the right mouse button
OnRightMouseDown	N/A	Executes a subroutine when the user clicks the scroll bar with the right mouse button
OnRightMouseUp	N/A	Executes a subroutine when the user releases the right mouse button while the pointer is on the scroll bar
PageNo	N/A	Specifies on which page of a multi-page form the scrollbar object appears
Parent	N/A	An object reference that points to the parent form
RangeMax	100.00	Determines the upper limit for the value linked to the scroll bar
RangeMin	1.00	Determines the lower limit for the value linked to the scroll bar
Release()	N/A	Removes the scroll bar definition from memory
SetFocus()	N/A	Gives focus to the scrollbar
StatusMessage	Empty string	Specifies a message to display on the status bar while the scroll bar has focus
TabStop	.T.	Determines if the user can give focus to the scroll bar by pressing <i>Tab</i> or <i>Shift+Tab</i>
Top	N/A	Sets the position of the top border
Value	N/A	Sets the value in the scroll bar
Vertical	.T.	Determines whether the scroll bar is aligned vertically or horizontally
Visible	.T.	Determines whether the scroll bar is visible or hidden
When	N/A	Specifies a condition that must evaluate to true before the user can give focus to the scroll bar
Width	N/A	Sets the width

Description

Use a scroll bar to let users vary numeric values rapidly. Unlike spin boxes, scroll bars don't accept keyboard input or use a Step value. Instead, the user drags the slider button to increase or decrease the value.

As the user moves the slider button, the value is continually updated to reflect the position of the button. For example, a scroll bar that varies a numeric value between 1 and 100 sets the value to 50 when the slider button is at the center of the scroll bar.

To set a range for the scrollbar, set RangeMin to the minimum value and RangeMax to the maximum value.

You can combine a scroll bar with an entry field so that values set through the scroll bar are reflected in the entry field. To do so, give the entry field the same DataLink specification as the scroll bar. Similarly, you can link a scroll bar to a numeric field with the DataLink property.

To automatically display the results of changes made using a datalinked scrollbar, assign a procedure or codeblock such as `{;form.refresh()}` to the scroll bar's `OnChange` property.

When you create a scroll bar with the `NEW` operator, you can specify two parameters:

- *<parent form reference>*—An object reference pointing to the parent form.
- *<object name expC>*—A character string assigned to the `Name` property of the new scroll bar. This value is optional.

For example, the following commands create a form and a scroll bar to display in it:

```
MyForm = NEW FORM()
MyScroll = NEW SCROLLBAR(MyForm, "OurScroll")
```

The `Name` property of the new scroll bar contains "OurScroll".

Example

The following example creates a form with an entry field object that displays the value in the `GNP` field of `Country.DBF`. A horizontal scroll bar can be used to change the field value within a specified range:

```
LOCAL f
f=NEW DISPLAY()
f.OPEN()
CLASS DISPLAY OF FORM
  this.Top=2
  this.Left=2
  this.Width=36
  this.Height=13
  this.View = "Country.DBF"
  this.Text= "Class Scrollbar Demo"
  DEFINE ENTRYFIELD Etf1 OF THIS;
    PROPERTY Datalink "Country->GNP",;
    Width 9, Top 4, Left 13
  DEFINE SCROLLBAR Sb1 OF THIS;
    PROPERTY Vertical .F., Height 2,;
    Top 6, Left 5,;
    Width 25, DataLink "Country->GNP",;
    RangeMin 1000, RangeMax 999999999
    OnChange {;form.refresh( )}
  DEFINE TEXT Txt1 OF THIS;
    PROPERTY Text "***Range: 1000 to 999999999***",;
    Top 9, Left 0, Width 36, Alignment 7
ENDCLASS
```

See Also

CLASS SPINBOX

CLASS SHAPE

A region of color within a form.

Properties

The following table lists the properties of the Shape class. For more information on each property, see Chapter 8.

Property	Default	Description
ClassName	SHAPE	Identifies the shape class
ColorNormal	BtnText/BtnFace	Sets the border and interior colors of the shape object
Height	N/A	Sets the height
Left	N/A	Sets the position of the left border
Move()	N/A	Moves or sizes the shape object
Name	SHAPE1	Specifies the name of the shape object
OnOpen	N/A	Executes a subroutine when the parent form is opened
PageNo	N/A	Specifies on which page of a multi-page form the shape object appears
Parent	N/A	An object reference that points to the parent form
PenStyle	0	Specifies one of a series of line styles to be used for the border of the shape object
PenWidth	1	Specifies the width of the border line of a shape object
Release()	N/A	Releases the shape object definition from memory
ShapeStyle	3 (Circle)	Specifies which of several styles are applied to a shape object
Top	N/A	Sets the position of the top border
Visible	.T.	Determines whether the shape object is visible or hidden
Width	N/A	Sets the width

Description

Use a Shape object to create a region of color within a form. The ShapeStyle property determines the shape of the region you create. Like a Line object, a Shape does not have an ID or an hWnd property.

While a Shape does not have a Border property, you can simulate a border by designating a pair of colors (<foreground color>/<background color>) for the ColorNormal property. The Shape object will display with a single-line border which is <foreground color> while the interior of the Shape object is <background color>. Setting ColorNormal to a single color value makes the entire shape that color.

Example

The following example creates a form and places an elliptical blue object with a bright white border inside the form.

```
MyForm = NEW FORM("Shape Display")
MyShape = NEW SHAPE(MyForm, "OURSHAPE")  &&Name property = "OURSHAPE"
MyShape.ShapeStyle = 2&& Elliptical shape
```

```
MyShape.ColorNormal = "W+/B"&& Bright white border, blue interior
MyForm.Open()
```

The Name property of the new Shape object contains "OURSHAPE".

See Also
CLASS IMAGE, CLASS RECTANGLE

CLASS SPINBOX

An object that lets the user enter values in a text box or change a value by clicking arrow buttons.

Properties
The following table lists the properties of the Spinbox class. For more information on each property, see Chapter 8.

Property	Default	Description
Before	N/A	Specifies which object the spin box precedes in the tabbing order of the parent form
Border	.T.	Determines if the spin box is surrounded with a border
ClassName	SPINBOX	Identifies the spin box class
ColorHighLight	WindowText /Window	Sets the color of the spin box when it's highlighted
ColorNormal	WindowText /Window	Sets the color of the spin box when it isn't highlighted
Copy()	N/A	Copies selected text to the Windows clipboard
Cut()	N/A	Cuts selected text and places it on the Windows clipboard
DataLink	Empty string	Links the spin box to a field
Enabled	.T.	Determines if the spin box can be selected
FontBold	.T.	Determines if characters are displayed in bold type
FontItalic	.F.	Determines if characters are displayed in italic type
FontName	MS Sans Serif	Specifies the font to apply to displayed characters
FontSize	N/A	Specifies the size of the font in points
FontStrikeOut	.F.	Determines if characters are displayed in strikeout type
FontUnderline	.F.	Determines if characters are displayed in underlined type
Function	Empty string	Formats displayed text
Height	N/A	Sets the height
HelpFile	Empty string	Identifies a Windows Help file (.HLP) that contains context-sensitive Help topics
HelpID	Empty string	Specifies the context string or context number of a Help topic in a Windows Help file (.HLP)
hWnd	N/A	Returns the object handle of the spin box
ID	-1	Identifies the spin box with a numeric value
Keyboard()	N/A	Passes a character string to the spin box, simulating typed user input

Property	Default	Description
Left	N/A	Sets the position of the left border
MousePointer	0	Specifies the mouse pointer type when the pointer is over the spin box
Move()	.F.	Moves or sizes the spin box
Name	SPINBOX1	Specifies the name of the spin box
OldStyle	.F.	Determines if the spin box is displayed in the default Windows style or in dBase style
OnChange	N/A	Executes a subroutine when the user changes a value
OnGotFocus	N/A	Executes a subroutine when the spin box receives focus
OnHelp	N/A	Executes a subroutine when the user presses <i>F1</i>
OnLeftDblClick	N/A	Executes a subroutine when the user double-clicks the spin box
OnLeftMouseDown	N/A	Executes a subroutine when the user clicks the spin box with the left mouse button
OnLeftMouseUp	N/A	Executes a subroutine when the user releases the left mouse button while the pointer is over the spin box
OnLostFocus	N/A	Executes a subroutine when focus is removed
OnMiddleDblClick	N/A	Executes a subroutine when the user double-clicks the spin box with the middle mouse button
OnMiddleMouseDown	N/A	Executes a subroutine when the user clicks the spin box with the middle mouse button
OnMiddleMouseUp	N/A	Executes a subroutine when the user releases the middle mouse button while the pointer is on the spin box
OnMouseMove	N/A	Executes a subroutine when the user moves the mouse in the spin box
OnOpen	N/A	Executes a subroutine when the parent form is opened
OnRightDblClick	N/A	Executes a subroutine when the user double-clicks the spin box with the right mouse button
OnRightMouseDown	N/A	Executes a subroutine when the user clicks the spin box with the right mouse button
OnRightMouseUp	N/A	Executes a subroutine when the user releases the right mouse button while the pointer is on the spin box
PageNo	N/A	Specifies on which page of a multi-page form the spin box object appears
Parent	N/A	An object reference that points to the parent form
Paste()	N/A	Copies text from the Windows clipboard to the current cursor position
Picture	Empty string	Formats text
RangeMax	100.00	Determines the upper limit for the value linked to the spin box
RangeMin	1.00	Determines the lower limit for the value linked to the spin box
RangeRequired	.F.	Determines whether the range you specify with RangeMax and RangeMin applies to all data(.T.) or to new or edited data only (.F.)
Release()	N/A	Removes the spin box definition from memory
SelectAll	.T.	Determines if the initial value is selected (highlighted) when the spin box receives focus
SetFocus()	N/A	Gives focus to the spin box
SpeedTip	Empty string	Specifies the text that appears when the mouse remains on the spin box for more than one second

Property	Default	Description
SpinOnly	.F.	Enables or disables editing in the text box portion of the spin box.
StatusMessage	Empty string	Specifies a message to display on the status bar while the spin box has focus
Step	1	Determines how much a user can add to and subtract from a value by clicking on the spin box button arrows
TabStop	.T.	Determines if the user can give focus to the spin box by pressing <i>Tab</i> or <i>Shift+Tab</i>
Top	N/A	Sets the position of the top border
Undo()	N/A	Reverses the effect of the most recent Cut(), Copy() or Paste() action
Valid	N/A	Specifies a condition that must evaluate to true (.T.) before the user can remove focus from the spin box
ValidErrorMsg	Empty string	Specifies a character string to display on the status bar when the Valid property returns false (.F.)
ValidRequired	.F.	Determines if the Valid property applies to all data or to new data only
Value	N/A	Sets the value in the spin box
Visible	.T.	Determines whether the spin box is visible or hidden
When	N/A	Specifies a condition that must evaluate to true before the user can give focus to the spin box
Width	N/A	Sets the width

Description

Use a spin box to let users enter values by typing them in the text box or by increasing or decreasing the current value using the up and down arrow buttons.

Spin boxes control the rate at which users change numeric or date values. For example, one spin box might change an interest rate in increments of hundredths, while another might change a date value in year increments. Set the size of each increment with the Step property; for example, if you set Step to 5, each click on an arrow changes a numeric value by 5 or a date value by 5 days.

Link a spin box to a numeric, float, or date field with the DataLink property. The Value property contains the current value of the field.

To restrict entries to those within a particular range of values, set the RangeMin property to the minimum value and RangeMax to the maximum value. If you want users to only select values with the arrow buttons, set SpinOnly to true (.T.).

When you create a spin box with the NEW operator, you can specify two parameters:

- *<parent form reference>*—An object reference pointing to the parent form.
- *<object name expC>*—A character string assigned to the Name property of the new spin box. This value is optional.

For example, the following commands create a form and a spin box to display in it:

```
MyForm = NEW FORM()
MySpin = NEW SPINBOX(MyForm, "OurSpin")
```

The Name property of the new spin box contains "OurSpin".

Example

The following example creates a form with a browse object displaying country population data and a spin box object for incrementing population by intervals of 1000:

```
LOCAL f
f=NEW DISPLAY()
f.OPEN()
CLASS DISPLAY OF FORM
  this.Top=2
  this.Left=2
  this.Width=36
  this.Height=13
  this.View = "Country.DBF"
  this.Text= "Class SpinBox Demo"
  DEFINE BROWSE BR1 OF THIS;
    PROPERTY Fields "Name, Population, GNP",;
    Top 1, Left 1.5, Width 32, Height 6
  DEFINE SPINBOX Spl OF THIS;
    PROPERTY;
    Datalink "Country->Population",;
    Top 10,;
    Left 19,;
    Height 2.0,;
    Width 15,;
    TabStop .F.,;
    Step 1000
  DEFINE TEXT Txt1 OF THIS;
    PROPERTY Top 10.5, Left 2, Width 16,;
    Text "Change Population:", ColorNormal "R/W"
ENDCLASS
```

See Also

CLASS SCROLLBAR

CLASS TABBOX

A class that makes available the type of tab-based control that is used in the SET dialog, the Property Inspector and other places in dBASE.

Properties

The following table lists the properties of the TabBox class. For more information on each property, see Chapter 8.

Property	Default	Description
Anchor	1	Specifies whether the tab box stays in the same relative position when the form is resized
Before	N/A	Specifies which object the tab box precedes in the tabbing order of the parent form

Property	Default	Description
ClassName	TABBOX	Identifies the tab box class
ColorHighlight	BtnText/BtnFace	Sets the color of the tab box when it is selected
ColorNormal	BtnText/BtnFace	Sets the color of the tab box when it isn't selected
CurSel	1	Specifies the currently-selected prompt in the tab box
DataSource	ARRAY {"TABBOX1"}	Specifies the prompts to display in the tab box
Enabled	.T.	Determines if the tab box can be selected
FontBold	.T.	Determines if characters in the tab box prompt are displayed in bold type
FontItalic	.F.	Determines if characters are displayed in italic type
FontName	MS Sans Serif	Specifies the font to apply to displayed characters
FontSize	N/A	Specifies the size of the font in points
FontStrikeOut	.F.	Determines if characters are displayed in strikeout type
FontUnderline	.F.	Determines if characters are displayed in underlined type
Height	1	Sets the height
HelpFile	Empty string	Identifies a Windows Help file (.HLP) that contains context-sensitive Help topics
HelpID	Empty string	Specifies the context string or context number of a Help topic in a Windows Help file (.HLP)
hWnd	N/A	Specifies the tab box handle
ID	100	Identifies the tab box with a numeric value
Left	0	Sets the position of the left border
Move()	N/A	Moves or sizes the tab box
Name	TABBOX1	Specifies the tab box name
OnGotFocus	N/A	Executes a subroutine when the tab box receives focus
OnHelp	N/A	Executes a subroutine when the user presses <i>F1</i>
OnLeftDblClick	N/A	Executes a subroutine when the user double-clicks the tab box
OnLeftMouseDown	N/A	Executes a subroutine when the user clicks the tab box with the left mouse button
OnLeftMouseUp	N/A	Executes a subroutine when the user releases the left mouse button while the pointer is over the tab box
OnLostFocus	N/A	Executes a subroutine when focus is removed
OnMiddleDblClick	N/A	Executes a subroutine when the user double-clicks the tab box with the middle mouse button
OnMiddleMouseDown	N/A	Executes a subroutine when the user clicks the tab box with the middle mouse button
OnMiddleMouseUp	N/A	Executes a subroutine when the user releases the middle mouse button while the pointer is on the tab box
OnMouseMove	N/A	Executes a subroutine when the user moves the mouse pointer over the tab box
OnOpen	N/A	Executes a subroutine when the parent form is opened
OnRightDblClick	N/A	Executes a subroutine when the user double-clicks the tab box with the right mouse button
OnRightMouseDown	N/A	Executes a subroutine when the user clicks the tab box with the right mouse button

Property	Default	Description
OnRightMouseUp	N/A	Executes a subroutine when the user releases the right mouse button while the pointer is on the tab box
OnSelChange	N/A	Executes a subroutine when the highlight is moved from one prompt to another
PageNo	0	Specifies on which page of a multi-page form the tab box object appears; a value of 0 means it appears on all pages
Parent	N/A	An object reference that points to the parent form
Release()	N/A	Removes the tab box definition from memory
SetFocus()	N/A	Gives focus to the tab box
TabStop	.T.	Determines if the user can give focus to the tab box by pressing <i>Tab</i> or <i>Shift+Tab</i>
Top	N/A	Sets the position of the top border
Visible	.T.	Determines whether the tab box is visible or hidden
When	N/A	Specifies a condition that must evaluate to true before the user can give focus to the tab box
Width	N/A	Sets the width

Description

A TabBox contains a number of tabs that users can select. The TabBox control behaves like a list box. As with the list box, you can use the CurSel and OnSelChange() properties to specify actions to perform.

By setting the PageNo property of a TabBox control to 0 (the default), you can implement a tabbed multi-page form where the user can easily switch pages by selecting tabs. Use the PageNo property of a control to determine on which page the control appears, and use the CurSel and OnSelChange() properties of the TabBox to switch between pages.

You can use the DataSource property with a literal array to specify the text prompts that appear on the tabs. For example, to create three tabs that say January, February, and March, use the following statement as the DataSource property:

```
Array {"January", "February", "March"}
```

Note There is a space between the word Array and the opening brace ({}). Tabs within a TabBox control are always situated horizontally.

Example

```
f = NEW Form()
DEFINE TABBOX t OF f;
PROPERTY;
    DataSource 'ARRAY {"Page 1", "Page 2", "Page 3"}'
f.Open()
```

See Also

CLASS ARRAY, CLASS LISTBOX

CLASS TEXT

A string of text characters.

Properties

The following table lists the properties of the Text class. For more information on each property, see Chapter 8.

Property	Default	Description
Alignment	0 (Top Left)	Positions text in the text object
Before	N/A	Specifies which object the text precedes in the tabbing order of the parent form
Border	.F.	Determines if the text object is surrounded with a border
ClassName	TEXT	Identifies the text class
ColorNormal	BtnText/ BtnFace	Sets the color of the text
FontBold	.T.	Determines if characters are displayed in bold type
FontItalic	.F.	Determines if characters are displayed in italic type
FontName	MS Sans Serif	Specifies the font to apply to displayed characters
FontSize	N/A	Specifies the size of the font in points
FontStrikeOut	.F.	Determines if characters are displayed in strikeout type
FontUnderline	.F.	Determines if characters are displayed in underlined type
Function	Empty string	Formats displayed text
GetTextExtent	N/A	Returns the length of the text object based on a comparison between the font of the text object and the font of the form
Height	1.00	Sets the height
hWnd	N/A	Returns the object handle of the text object
ID	-1	Identifies the text object with a numeric value
Left	N/A	Sets the position of the left border
MousePointer	0	Specifies the mouse pointer type when the pointer is over the text object
Move()	N/A	Moves or sizes the text object
Name	TEXT1	Specifies the name of the text object
OldStyle	.F.	Determines if the text object is displayed in the default Windows style or in dBase style
OnLeftDblClick	N/A	Executes a subroutine when the user double-clicks the text object
OnLeftMouseDown	N/A	Executes a subroutine when the user clicks the text object with the left mouse button
OnLeftMouseUp	N/A	Executes a subroutine when the user releases the left mouse button while the pointer is over the text object
OnMiddleDblClick	N/A	Executes a subroutine when the user double-clicks the text object with the middle mouse button
OnMiddleMouseDown	N/A	Executes a subroutine when the user clicks the text object with the middle mouse button
OnMiddleMouseUp	N/A	Executes a subroutine when the user releases the middle mouse button while the pointer is on the text object

Property	Default	Description
OnMouseMove	N/A	Executes a subroutine when the user moves the mouse in the text object
OnOpen	N/A	Executes a subroutine when the parent form is opened
OnRightDbClick	N/A	Executes a subroutine when the user double-clicks the text object with the right mouse button
OnRightMouseDown	N/A	Executes a subroutine when the user clicks the text object with the right mouse button
OnRightMouseUp	N/A	Executes a subroutine when the user releases the right mouse button while the pointer is on the text object
PageNo	N/A	Specifies on which page of a multi-page form the text object appears
Parent	N/A	An object reference that points to the parent form
Picture	Empty string	Formats text
Release()	N/A	Removes the text object definition from memory
Text	N/A	Specifies the displayed character string
Top	N/A	Sets the position of the top border
Visible	.T.	Determines whether the text object is visible or hidden
Width	N/A	Sets the width

Description

Use a text object as a prompt, heading, label, or reminder. For example, a text object can prompt the user to enter values in an entry field, or give brief instructions on using a scroll bar.

Use the Text property to specify the character string to display. The displayed text is read-only; the user can't give it focus or modify it directly.

When you create a text object with the NEW operator, you can specify two parameters:

- *<parent form reference>*—An object reference pointing to the parent form.
- *<object name expC>*—A character string assigned to the Name property of the new text object. This value is optional.

For example, the following commands create a form and a text object to display in it:

```
MyForm = NEW FORM()
MyText = NEW TEXT(MyForm, "OurText")
```

The Name property of the new text object contains "OurText".

Example

The following example places three text line samples on the form. The first uses a font predefined by #define; the second and third demonstrate alternative parameter options:

```
#define BIGFONT    FontName "Arial", FontSize 26
LOCAL f
f=NEW Demo()
f.OPEN()
CLASS Demo OF FORM
    this.Top=2
```

CLASS TEXT

```
this.Left=2
this.Width=50
this.Height=13
this.Text= "Class TEXT Demo"
DEFINE TEXT Txt1 OF THIS;
  PROPERTY;
    Top 1;;
    Left 0;;
    Alignment 4;;
    Text "Visual dBASE 5.7";
    Height 3;;
    Width 50;;
    BIGFONT;;
    ColorNormal "R/W"
DEFINE TEXT Txt2 OF THIS;
  PROPERTY;
    Top 5;;
    Left 0;;
    Alignment 4;;
    Text "is here in '99";
    Height 2;;
    Width 50;;
    FontBold .T.;
    FontItalic .T.;
    FontSize 16
DEFINE TEXT Txt3 OF THIS;
  PROPERTY;
    Top 8;;
    Left 10;;
    Alignment 4;;
    Text "from dBASE Inc.";
    Height 3;;
    Width 30;;
    FontBold .T.;
    FontSize 22;;
    Border .T.;
    ColorNormal "GB/W"
ENDCLASS
```

See Also
DEFINE



Properties

Properties

AbandonRecord()

Releases a newly-created record from memory.

Property of class

FORM

Description

Use `AbandonRecord()` to cancel the creation of a new record stored in a temporary memory buffer you created with `BeginAppend()`.

For more information, see `BeginAppend()`.

Example

See `BeginAppend()` for an example.

See Also

`BeginAppend()`, `IsRecordChanged()`, `SaveRecord()`

ActiveControl

Contains a reference to the object that currently has focus.

Property of class

FORM

Data type

Object reference

Description

Use ActiveControl property to reference the object that currently has focus.

An object gets focus in three ways:

- The user tabs to the object.
- The user clicks the object.
- The SetFocus() method of the object is executed.

Use ActiveControl to find out which object currently has focus and make branching decisions accordingly. For example, the following command looks at the Name property of the current object:

```
? MyForm.ActiveControl.Name
```

You can also use ActiveControl to access or change a property of the object with focus. For example, the following command disables whatever object currently has focus:

```
MyForm.ActiveControl.Enabled = .F.
```

Example

The following example uses ActiveControl and NextObj to return the name of the current and next entry field objects when the right mouse is clicked on a form. This would be useful on a form where entry fields had no accompanying text to identify the fields:

```
LOCAL f
f=NEW EntryForm()
f.OPEN()
CLASS Entryform OF FORM
  this.View="Company.DBF"
  this.OnRightMouseDown={;Fcs="Current Field: " + ;
    Form.ActiveControl.Name; Form.Txt1.Text=Fcs}
  this.OnRightMouseUp={;Fcs2="Next Field:      " + ;
    Form.NextObj.Name; Form.Txt2.Text=Fcs2}
DEFINE ENTRYFIELD Company OF THIS;
  PROPERTY Datalink "Company->Company",;
  Top 3, Left 1, Width 20, Name "Company"
DEFINE ENTRYFIELD State OF THIS;
  PROPERTY Datalink "Company->State_Prov",;
  Top 5, Left 1, Name "State"
DEFINE TEXT Txt1 OF THIS;
  PROPERTY Text " ",;
  Top 8, Left 1, Width 25
DEFINE TEXT Txt2 OF THIS;
  PROPERTY Text " ",;
  Top 9, Left 1, Width 25
ENDCLASS
```

See Also

[_curobj](#), [First](#), [ID](#), [NextObj](#)

Add()

Adds an element to a one-dimensional array object.

Property of class

ARRAY

Description

Use the Add() method to add a single element to a one-dimensional array object.

Add() requires *<expN>*, a parameter that specifies what value to put in the new element. For example, an array object with 10 elements acquires an eleventh element containing 100 when *<expN>* is 100, as with:

```
MyArray = NEW ARRAY(10)
MyArray.Add(100)
```

See Insert() for another way to add elements to an array object.

Example

```
USE Customer.DBF
* Initialize an array object
ObjArr=NEW ARRAY(RECCOUNT())
* Fill 1-dimensional array with values
* from Name field of Customer.DBF
GO TOP
FOR i=1 TO RECCOUNT()
    ObjArr[i]= Customer->Name
    SKIP
NEXT i
* Use ADD() to add an additional element
* to the 1-dimensional array and place
* a customer name in that element.
ObjArr.ADD(1)
Cnt=ObjArr.SIZE      && Returns new number of elements
ObjArr[Cnt] = "George Benson Dive Works"
* Display Contents array
FOR i=1 TO Cnt
    ? ObjArr[i] AT 10
NEXT i
```

See Also

Grow(), Insert(), Resize()

Advise()

Requests that a server application notify dBASE when an item in a server topic changes.

Property of class

DDELINK

Description

Use the Advise() method to create a *hot link* to an item in a server topic. A hot link makes the server notify dBASE when the item changes.

A server topic can be anything that the server application understands, but it is usually a document you open in the external application. For example, a data-exchange program might start a session in Quattro Pro for Windows, open one of its spreadsheet files (the topic), and use Advise() to establish a hot link to one of its cells.

Advise() requires the *<item>* parameter, which identifies the hot-linked item in the server topic. This item can be any single element, such as a field in a table or a cell in a spreadsheet. For example, you can specify cell C2 of Page A in a Quattro Pro spreadsheet file by passing the parameter "A:C2". When the contents of the cell are altered, Quattro Pro notifies dBASE and a subroutine (which you specify with the OnNewValue property) is executed automatically.

Example

```
PUBLIC LinkObj
LinkObj = NEW DDELINK()
LinkObj.OnNewValue = Valuehandler;
                        && Codeblock or function pointer
LinkObj.Initiate("QPW", "Demo.WB1")
LinkObj.Advise("A:A1");
                        && Notified when cell A:A1 changes
```

See Also

Execute(), Initiate(), OnNewValue, Peek(), Poke(), Server, Terminate(), Timeout, Topic, Unadvise()

Alias

Determines the table that is accessed by a browse object.

Property of class

BROWSE

Data type

Character

Default

The default for Alias is an empty string.

Description

Use the Alias property to identify a table to display in a browse object.

An *alias* is an alternate name given to an open table file, and can consist of

- A name you specify with the ALIAS option of the USE command.
- The file name of the table (if you did not assign the table an alias).
- The letter that corresponds to the work area of the table. This alias can be any letter from A to J.

For example, when the parent form is based on a query that opens two or more files in a parent-child relation, you can use Alias to determine which table appears in the browse object.

For more information on aliases and work areas, see SELECT and USE.

Example

NEW operator syntax:

```
this.Alias = "Animals"
```

DEFINE object syntax:

```
DEFINE BROWSE Br1 OF THIS;
    PROPERTY Alias "Animals"
```

See Also

DataLink, DEFINE

Alignment

Positions a graphic in an image object or text in a text object.

Property of class

IMAGE, TEXT

Data type

Numeric

Default

The default for Alignment is 0 (Top Left).

Description

Use Alignment when an image object or a text object is larger than the image or text it contains.

For example, if you store graphics in a binary field and link this field to an image object through the DataSource property, the user can display each image by moving from record to record. If the images vary in size, the smaller images might not fill the entire object. Use alignment to position the images within the object.

You can assign the following settings to the Alignment property of a text object:

Setting	Description
0 (Top Left)	Adjacent to the top edge and the left edge
1 (Top Center)	Adjacent to the top edge and centered horizontally
2 (Top Right)	Adjacent to the top edge and the right edge
3 (Center Left)	Centered vertically and adjacent to left edge
4 (Center)	Centered horizontally and vertically
5 (Center Right)	Centered vertically and adjacent to right edge
6 (Bottom Left)	Adjacent to the bottom edge and the left edge
7 (Bottom Center)	Centered horizontally and adjacent to the bottom edge
8 (Bottom Right)	Adjacent to the bottom edge and the right edge
9 (Wrap Left)	Wraps strings that exceed the width of the text object

You can assign the following settings to the Alignment property of an image object:

Setting	Description
0 (Stretch)	Enlarged to fill the entire image object
1 (Top Left)	Adjacent to the left edge and the top edge
2 (Center)	Centered
3 (Keep Aspect Stretch)	Maintains the original height/width ratio when the image is displayed or resized

Note Specify the dimensions of a text object or an image object with the Height and Width properties.

Example

NEW operator syntax:

```
Heading = NEW Text(this)
Heading.Text = "Animals of the World"
Heading.Alignment = 7
Heading.Border = .T.
```

DEFINE object syntax:

```
DEFINE TEXT Heading OF THIS;
PROPERTY;
Text "Animals of the World",;
Alignment 7,;
Border .T.
```

See Also

Height, Width

Anchor

Specifies whether an object stays in the same relative position when the form is resized.

Property of class

TABBOX

Data type

Numeric

Default

The default for Anchor is 1 (Bottom).

Description

Use Anchor to specify whether a tab box should maintain its size and location even if the parent form is resized. Acceptable values for Anchor are 1 (Bottom) and 0 (None). Generally, you'll want tabs on a form to automatically resize and reposition themselves as their parent form is resized, so you'll want to set Anchor to 1. For example, if you are using a tab box to move between different pages in a form, one tab is equivalent to one page, so the size of the tab and the size of the page (or form) should be the same.

However, if you want the tabs to retain a particular placement and size configuration despite the size of the parent form, set Anchor to 0.

Example

```
f = NEW Form()
DEFINE TABBOX TABBOX1 OF f PROPERTY Anchor 1
```

See Also

PageCount(), PageNo

Append

Determines if records can be added to a table in a browse object.

Property of class

BROWSE

Data type

Logical

Default

The default for Append is true (.T.).

Description

Set Append to false (.F.) when you want to prevent users from adding records to a table. For example, an application might allow executives to view or even modify existing customer accounts, without letting them add new accounts.

Example

NEW operator syntax:

```
CompanyBrowse = New BROWSE(this)
CompanyBrowse.Top = 3
CompanyBrowse.Left = 1
CompanyBrowse.Width = 60
CompanyBrowse.Alias = "Company"
CompanyBrowse.Append = .F.
CompanyBrowse.Delete = .F.
CompanyBrowse.Modify = .F.
```

DEFINE object syntax:

```
DEFINE BROWSE CompanyBrowse OF THIS;
FROM 3,1 TO 13,40;
Property;
Alias "Company",;
Append .F. ,;
Delete .F. ,;
Modify .F.
```

See Also

Delete, Modify

AutoSize

Determines if a form is automatically sized to contain its objects when the form is opened.

Property of class

FORM

Data type

Logical

Default

The default for AutoSize is false (.F.).

Description

Use AutoSize to determine how a form is sized and proportioned.

If you set the `AutoSize` property of a form to `true` (.T.), the form is automatically adjusted to contain its objects when it is opened. If you set `AutoSize` to `false` (.F.), the form assumes its default dimensions when it is opened.

The default dimensions of a form are determined by the following:

- The settings you give to the `Height` and `Width` properties
- The `FROM...TO` clause of the `DEFINE FORM` command, if you used that command to create the form
- The size and proportion the user last gave to the form with the mouse or the `Size` option of the `Control Menu`

When you set the `AutoSize` property of a form to `true`, the default dimensions are ignored. The user can still move or resize the form, but if the form is closed and reopened it is automatically resized again to contain its objects.

Example

`NEW` operator syntax:

```
EntryForm = NEW Form()
EntryForm.AutoSize = .F.
```

`DEFINE` object syntax:

```
DEFINE FORM EntryForm ;
Property;
AutoSize .F.
```

See Also

`Height`, `Left`, `Width`

Before

Specifies which other object an object precedes in the tabbing order of the form.

Property of class

`BROWSE`, `CHECKBOX`, `COMBOBOX`, `EDITOR`, `ENTRYFIELD`, `IMAGE`, `LINE`, `LISTBOX`, `MENU`, `OLE`, `PAINTBOX`, `PUSHBUTTON`, `RADIOBUTTON`, `RECTANGLE`, `SCROLLBAR`, `SPINBOX`, `TABBOX`, `TEXT`

Data type

Object reference

Description

Use the `Before` property to control the tabbing order of the objects in a form.

When a form holds two or more objects that users can select, the user can move from object to object by pressing *Tab* or *Shift+Tab*. The order in which focus moves from object to object is known as the *tabbing order*.

BeginAppend()

When you insert an object reference that points to one object (Object A) into the Before property of another object (Object B), the tabbing order of the form is changed so that

- When the user presses *Tab* to move from object to object, Object B precedes Object A.
- When the user presses *Shift+Tab* to move from object to object, Object A precedes Object B.

Example

NEW operator syntax:

```
LOCAL f
f=NEW EntryForm()
f.OPEN()
CLASS Entryform OF FORM
    this.View="Company.DBF"
    this.Fld1 = NEW Entryfield(this)
    this.Fld1.Top = 3
    this.Fld1.Left = 1
    this.Fld1.Width=20
    this.Fld1.Datalink = "Company->Company"
    this.Fld2 = NEW Entryfield(this)
    this.Fld2.Top = 5
    this.Fld2.Left = 1
    this.Fld2.Datalink = "Company->State_Prov"
    this.Fld2.Before = this.Fld1
ENDCLASS
```

DEFINE object syntax:

```
DEFINE ENTRYFIELD Company OF THIS;
    Property Datalink "Company->Company",;
    Top 3, Left 1, Width 20
DEFINE ENTRYFIELD State OF THIS;

    Property Datalink "Company->State_Prov",;
    Top 5, Left 1, Before this.Company
```

See Also

_curobj, CUATab

BeginAppend()

Creates a temporary buffer in memory for a record that is based on the structure of the current table, letting the user input data to the record without automatically adding the record to the table.

Property of class

FORM

Description

BeginAppend() creates a single record buffer in the current table, without actually adding the record to the table until SaveRecord() is issued. While this buffer exists, the user can input data to the record with controls such as an entry field or a check box. Use SaveRecord() to append the record to the currently active table, and use AbandonRecord() to discard the record. Use IsRecordChanged() to determine if the record has been changed since the BeginAppend() was issued.

For example, a form might contain two pushbuttons, one labeled Save and the other labeled Abandon. The OnClick subroutine of the Save pushbutton might execute SaveRecord() and the OnClick subroutine of the Abandon pushbutton might execute AbandonRecord().

You might also attach a procedure to the Abandon pushbutton to check the status of IsRecordChanged(). If it is true, you could ask the user to confirm that they want to cancel the append operation.

Using BeginAppend() has different results than using either BEGINTRANS() and APPEND BLANK or APPEND AUTOMEM. With these commands, if you cancel the append operation, you have a record marked for deletion added to the table. If you use AbandonRecord() to cancel the BeginAppend() operation, a new record is never added to the table.

Example

```
local f
f = new ANIFORM()
f.Open()
CLASS ANIFORM OF FORM
  this.Top = 3.5879
  this.PageNo = 1
  this.Width = 57.666
  this.ColorNormal = "N/BTNFACE"
  this.View = "animals.dbf"
  this.Text = "Form"
  this.TopMost = .F.
  this.ScrollBar = 2
  this.Height = 10.9404
  this.Left = 24
  this.OnOpen = CLASS::FORM_ONOPEN
  DEFINE TEXT TITLE OF THIS;
    PROPERTY;
      Top 0.5;;
      PageNo 1;;
      Width 28;;
      FontSize 18;;
      ColorNormal "HIGHLIGHT/BTNFACE",;
      Text "Animals",;
      Border .F.,;
      Height 2.0293;;
      Left 1
  DEFINE TEXT TEXT1 OF THIS;
    PROPERTY;
      Top 3,;
```

BeginAppend()

```
PageNo 1;;
Width 14;;
ColorNormal "BTNTEXT/BTNFACE",;
Text "&Name",;
Border .T.,;
OldStyle .T.,;
Height 2;;
Left 1
DEFINE ENTRYFIELD ENTRYFIELD1 OF THIS;
PROPERTY;
ColorHighLight "WindowText/Window",;
Top 4;;
PageNo 1;;
Width 13.3672;;
ColorNormal "WINDOWTEXT/WINDOW",;
Border .F.,;
Height 0.8232;;
Left 1.2988;;
DataLink "ANIMALS->NAME"
DEFINE TEXT TEXT2 OF THIS;
PROPERTY;
Top 3;;
PageNo 1;;
Width 14;;
ColorNormal "BTNTEXT/BTNFACE",;
Text "&Size",;
Border .T.,;
OldStyle .T.,;
Height 2;;
Left 15
DEFINE SPINBOX SPINBOX1 OF THIS;
PROPERTY;
ColorHighLight "WindowText/Window",;
Top 4;;
PageNo 1;;
Width 13.3672;;
ColorNormal "WINDOWTEXT/WINDOW",;
Border .F.,;
Rangemax 100;;
Rangemin 1;;
Height 0.8232;;
Left 15.2988;;
DataLink "ANIMALS->SIZE"
DEFINE TEXT TEXT3 OF THIS;
PROPERTY;
Top 3;;
PageNo 1;;
Width 28;;
ColorNormal "BTNTEXT/BTNFACE",;
Text "We&ight",;
Border .T.,;
OldStyle .T.,;
Height 2;;
Left 29
DEFINE SPINBOX SPINBOX2 OF THIS;
```

```

PROPERTY;
    ColorHighLight "WindowText/Window",;
    Top 4,;
    PageNo 1,;
    Width 13.3672,;
    ColorNormal "WINDOWTEXT/WINDOW",;
    Border .F.,;
    Rangemax 100,;
    Rangemin 1,;
    Height 0.8232,;
    Left 29.2988,;
    DataLink "ANIMALS->WEIGHT"
DEFINE TEXT TEXT4 OF THIS;
PROPERTY;
    Top 5,;
    PageNo 1,;
    Width 56,;
    ColorNormal "BTNTEXT/BTNFACE",;
    Text "&Area",;
    Border .T.,;
    OldStyle .T.,;
    Height 2,;
    Left 1
DEFINE ENTRYFIELD ENTRYFIELD2 OF THIS;
PROPERTY;
    ColorHighLight "WindowText/Window",;
    Top 6,;
    PageNo 1,;
    Width 20.0342,;
    ColorNormal "WINDOWTEXT/WINDOW",;
    Border .F.,;
    Height 0.8232,;
    Left 1.2988,;
    DataLink "ANIMALS->AREA"
DEFINE PUSHBUTTON PUSHBUTTON1 OF THIS;
PROPERTY;
    Top 8,;
    PageNo 1,;
    Width 14.167,;
    ColorNormal "BtnText/BtnFace",;
    Text "&Save",;
    Default .T.,;
    OnClick CLASS::PUSHBUTTON1_ONCLICK,;
    UpBitmap "RESOURCE #20",;
    DownBitmap "RESOURCE #20",;
    DisabledBitmap "RESOURCE #21",;
    FocusBitmap "RESOURCE #20",;
    Group .T.,;
    Height 1.8818,;
    Left 9.166
DEFINE PUSHBUTTON PUSHBUTTON2 OF THIS;
PROPERTY;
    Top 8,;
    PageNo 1,;
    Width 14.167,;

```

Border

```
ColorNormal "BtnText/BtnFace",;
Text "&Abandon",;
OnClick CLASS::PUSHBUTTON2_ONCLICK,;
UpBitmap "RESOURCE #24",;
DownBitmap "RESOURCE #24",;
DisabledBitmap "RESOURCE #25",;
FocusBitmap "RESOURCE #24",;
Group .T.,;
Height 1.8818,;
Left 31.166
Procedure PUSHBUTTON1_OnClick
  IF Form.IsRecordChanged()
    Form.SaveRecord()
  ENDF
  GOTO BOTTOM
  SKIP
  Form.BeginAppend()
RETURN

Procedure PUSHBUTTON2_OnClick
  Form.AbandonRecord()
  Form.Close()
RETURN

Procedure Form_OnOpen
  GOTO BOTTOM
  SKIP
  This.BeginAppend()
RETURN

ENDCLASS
```

See Also

AbandonRecord(), APPEND AUTOMEM, BEGINTRANS(), IsRecordChanged(), SaveRecord()

Border

Determines if an object is surrounded with a border.

Property of class

EDITOR, ENTRYFIELD, OLE, RECTANGLE, SPINBOX, TEXT

Data type

Logical

Default

The default for Border is different for different types of objects. See the individual CLASS statement for an object to determine the default for that object.

Description

You can prevent a rectangle object from having a border by setting the Border property to false (.F.). To restore the border, set the Border property to true.

By default, the border is a single line that surrounds the object. If you set a related property named BorderStyle to 1 (Raised), the object appears elevated, and if you set BorderStyle to 2 (Lowered), the object appears lowered.

Example

NEW operator syntax:

```
Heading = NEW TEXT(this)
Text = "Data Entry Screen"
Top = 1
Left = 5
Border = .T.
```

DEFINE object syntax:

```
DEFINE TEXT Heading OF EntryForm;
Property;
Text "Data Entry Screen",;
Top 0,;
Left 5,;
Border .T.
```

See Also

CLASS RECTANGLE, BorderStyle

BorderStyle

Determines the border characteristics of a rectangle object.

Property of class

RECTANGLE

Data type

Numeric

Default

The default for BorderStyle is 0 (Normal).

Description

Use `BorderStyle` to set the appearance of a rectangle object.

BorderStyle	Result
0 (Normal)	Rectangle is surrounded by a 3-D line with a title.
1 (Raised)	Rectangle appears elevated.
2 (Lowered)	Rectangle appears lowered.

When you set `BorderStyle` to 1 or 2, the label that is displayed in the top left corner of the rectangle disappears.

Example

NEW operator syntax:

```
Bx1 = NEW RECTANGLE(this)
  Top  = 1
  Left = 5
  Width = 30
  Height = 15
  BorderStyle = 2    && Lowered border style
```

DEFINE object syntax:

```
DEFINE RECTANGLE Bx1 OF THIS;
  Property;
  Top  1;;
  Left 5;;
  Width 30;;
  Height 15;;
  BorderStyle 1    && Raised border style
```

See Also

Border, CLASS RECTANGLE

Bottom

Sets the row position of the lower end of a line object.

Property of class

LINE

Data type

Numeric

Description

Use the `Bottom` property in combination with the `Right`, `Left`, and `Top` properties to determine the position and length of a line object.

Each unit of the value you assign to Bottom is the average height of characters in the active font of the parent form. For example, if you set the Bottom property of a line object to 17.5, the lower end of the line is positioned 17.5 characters down.

Example

```
DEFINE LINE Ln1 OF THIS;
PROPERTY;
  Left 10;;          && Vertical line from 3,10;
  Top 3;;            to 8,10
  Width 4;;
  Bottom 8;;
  ColorNormal "RB/W"
DEFINE LINE Ln2 OF THIS;
PROPERTY;
  Left 3;;          && Horizontal line from 8,3
  Top 8;;           8,33
  Width 4;;
  Bottom 8;;
  Right 33;;
  ColorNormal "RB/W"
```

See Also

CLASS FORM, Right, ScaleFontName, ScaleFontSize, Top, Width

CanClose

Executes a subroutine that determines if a form can be closed when an attempt is made to close the form.

Property of class

FORM

Description

Use CanClose to prevent a form from closing until certain conditions are met. CanClose can also be used to perform tasks when the form is about to close, but the form and its objects are still in scope. This is different from OnClose which does not execute its subroutine until the form is closed and the form's object are out of scope. The subroutine you assign to CanClose returns a value of true (.T.) which allows the form to close, or false (.F.) which prevents the form from closing.

For example, when a form is based on a QBE that joins two tables on a key field, you might want to prevent the key field of the parent table from containing blank or duplicated values. The subroutine you assign to CanClose might search the parent table for blank or duplicated values and return true (.T.) if no such values exist (allowing the form to close) or return false (.F.) if such values do exist (preventing the form from closing).

Example

```

f = new SAMPLEFORM()
f.Open()
CLASS SAMPLEFORM OF FORM
  * Form can only be closed if entryfield is not blank
  this.CanClose = CLASS::FORM_CANCLOSE
  DEFINE ENTRYFIELD ENTRYFIELD1 OF THIS;
    PROPERTY;
      Value "          ";
      Top 2;;
      Width 10;;
      Height 1;;
      Left 4
  * Returns .T. only if TestField is not blank
  Procedure Form_CanClose
    RETURN IIF(ISBLANK(TRIM(THIS.ENTRYFIELD1.VALUE)), .F., .T.)
ENDCLASS

```

See Also

Close(), OnClose

Checked

Determines if a checkmark appears beside a menu command.

Property of class

MENU

Data type

Logical

Default

The default for Checked is false (.F.).

Description

Use Checked to indicate that a condition or a process is turned on or off.

The checkmark appears to the left of the menu command when you set the Checked property to true (.T.); the checkmark is removed when you set the Checked property to false. The user usually adds or removes a checkmark by clicking the menu item.

You can query the current Checked setting and make branching decisions accordingly. For example, if Checked is true, you can execute a procedure or a codeblock; and if Checked is false, you can execute a different procedure or codeblock (or none at all).

Example**NEW operator syntax:**

```
Edit = NEW MENU(this.Main.View)
Edit.Text = "Edit"
Edit.Checked = .T.
```

DEFINE object syntax:

```
DEFINE MENU Edit OF THIS;
PROPERTY ;
Text "Edit",;
Checked .T.
```

See Also

DEFINE

ClassName

A read-only property that identifies the class that the object is based on.

Property of class

ARRAY, ASSOCARRAY, BROWSE, CHECKBOX, COMBOBOX, DDELINK, DDETOPIC, EDITOR, ENTRYFIELD, FORM, IMAGE, LINE, LISTBOX, MENU, MENUBAR, OLE, PAINTBOX, POPUP, PUSHBUTTON, RADIOBUTTON, RECTANGLE, SCROLLBAR, SHAPE, SPINBOX, TABBOX, TEXT

Data type

Character

Default

The name of the class the object is based on.

Description

Use ClassName to find out which class an object was created from. The ClassName value is set when you create the object, and is read-only.

Example**NEW operator syntax:**

```
this.Fld1 = NEW ENTRYFIELD(this)
this.Fld1.OnLostFocus = CheckIt
```

DEFINE object syntax:

```
DEFINE ENTRYFIELD Fld1 OF THIS;
PROPERTY OnLostFocus CheckIt
PROCEDURE CheckIt
IF This.Classname = "ENTRYFIELD"
? this.Value      && contents to Command window
ENDIF             && results pane or printer
```

See Also

Name

Close()

Closes a form.

Property of class

FORM

Description

Use Close() to close an open form.

When you try to close a form, dBASE does the following:

- 1 Executes the Valid subroutine or codeblock of the current object. If it returns a value of false (.F.), the form doesn't close.
- 2 Executes the OnLostFocus subroutine or codeblock of the current object.
- 3 Executes the CanClose subroutine or codeblock of the form. If it returns a value of false (.F.), the form doesn't close.
- 4 Executes the OnLostFocus subroutine or codeblock of the form.
- 5 Removes the form and the objects it contains from the screen.
- 6 Executes the OnClose subroutine or codeblock of the form.
- 7 Removes the form definition from memory if there are no object references pointing to the form.

If the form definition is not removed from memory, you can open the form again with Open() or OPEN FORM.

Example

NEW operator syntax:

```
this.Exit = NEW PUSHBUTTON(this)
this.Top=16
this.Left=14
this.Text="Exit"
this.Height=2
this.OnClick {;Form.Close()}
```

DEFINE object syntax:

```
DEFINE PUSHBUTTON Exit OF THIS;
PROPERTY Text "Exit", Height 2,;
Top 16, Left 14,;
OnClick {;Form.Close()}
```

See Also

CANCLOSE, CLOSE FORMS, Open(), OPEN FORM, ReadModal(), READMODAL()

ColorHighlight

Sets the color of the object that has focus.

Property of class

BROWSE, ENTRYFIELD, LISTBOX, SPINBOX, TABBOX

Data type

Character

Default

The default for ColorHighlight is different for different types of objects. See the individual CLASS statement for an object to determine the default for that object.

Description

Use the ColorHighlight and ColorNormal properties of an object so that users can differentiate visually when an object has focus and when it doesn't. For more information, see ColorNormal.

Example

See Datalink for an example of ColorHighlight.

See Also

ColorNormal, DEFINE

ColorNormal

Sets the color of an object that is not currently selected.

Property of class

BROWSE, CHECKBOX, COMBOBOX, EDITOR, ENTRYFIELD, FORM, LINE, LISTBOX, PAINTBOX, PUSHBUTTON, RADIOBUTTON, RECTANGLE, SCROLLBAR, SHAPE, SPINBOX, TABBOX, TEXT

Data type

Character

Default

The default for ColorNormal is different for different types of objects. See the individual CLASS statement for an object to determine the default for that object.

Description

Use the ColorNormal and ColorHighlight properties of an object so that users can differentiate visually when an object has focus and when it doesn't.

You can specify two color settings with ColorNormal: a foreground color (for text), and a background color. Color settings must be separated with a forward slash (/).

Default color settings are taken from the settings in the Windows Control Panel. If the colors are changed in the Control Panel while a form is open, the form and any controls that use these values will change automatically. You can use any of the following Windows color settings for either the foreground or background color:

ActiveBorder	BtnText	InactiveCaptionText
ActiveCaption	CaptionText	Menu
AppWorkspace	GrayText	MenuText
Background	Highlight	Scrollbar
BtnFace	HighlightText	Window
BtnHighlight	InactiveBorder	WindowFrame
BtnShadow	InactiveCaption	WindowText

You can also use the following color codes:

Black	N	Magenta	RB
Blue	B	Brown	RG
Green	G	White	W
Cyan	GB	Blank	X
Red	R		

These settings have no effect on monochrome monitors.

If you use color codes, you can also specify intensity (brightness) attributes. For both color and monochrome monitors, the attribute code for high-intensity foreground is +, and the tribute code for high-intensity background is *. For example, if you want the text in an entry field to appear in bright blue over a white background, set the ColorNormal property to B+ /W.

For monochrome monitors, I is an alternate attribute code for high intensity.

Note You can specify a setting for ColorNormal with the Choose Color dialog box, in which you choose a predefined color or design a color of your own. To access the Choose Color dialog box, click on the Tool button next to the ColorNormal item in the Inspector.

Example

See Datalink for an example of ColorNormal.

See Also

ColorHighlight, DEFINE

Copy()

Copies selected text to the Windows clipboard.

C

Property of class

BROWSE, COMBOBOX, EDITOR, ENTRYFIELD, SPINBOX

Description

Use Copy() when the user has selected text and wants to copy it to the Windows clipboard. The action of Copy() is identical to the Copy menu item on the standard Windows Edit menu.

If you have assigned a menubar to the form, you can use the menubar's EditCopyMenu property instead of using the Copy() property of individual objects on the form. For more information, see EditCopyMenu.

Example

```
local f
f = new SAMPLEFORM()
f.Open()
CLASS SAMPLEFORM OF FORM
  this.Width = 51.833
  this.Text = "Sample Form"
  this.Height = 12.6465
  this.OnOpen = CLASS::FORM_ONOPEN
  DEFINE ENTRYFIELD ENTRYFIELD1 OF THIS;
    PROPERTY;
    OnGotFocus CLASS::ENTRYFIELD1_ONGOTFOCUS;;
    Value "Sample Text",;
    Border .T.,;
    Top 3,,;
    PageNo 1,,;
    Width 25,,;
    ColorNormal "WINDOWTEXT/WINDOW",;
    Height 1.5,,;
    ColorHighLight "WINDOWTEXT/WINDOW",;
    Left 3
  DEFINE PUSHBUTTON PUSHBUTTON1 OF THIS;
    PROPERTY;
    Group .T.,;
    Top 1,,;
    PageNo 1,,;
    Width 14,,;
    ColorNormal "BtnText/BtnFace",;
    Text "&Undo",;
    OnClick CLASS::PUSHBUTTON1_ONCLICK,,;
    Height 1.5,,;
    Enabled .F.,;
    Left 33.5
  DEFINE PUSHBUTTON PUSHBUTTON2 OF THIS;
    PROPERTY;
```

Copy()

```
    Group .T.,;
    Top 4,,;
    PageNo 1,,;
    Width 14,,;
    ColorNormal "BtnText/BtnFace",;
    Text "Cu&t",;
    OnClick CLASS::PUSHBUTTON2_ONCLICK,,;
    Height 1.5,,;
    Left 33.5
DEFINE PUSHBUTTON PUSHBUTTON3 OF THIS;
PROPERTY;
    Group .T.,;
    Top 7,,;
    PageNo 1,,;
    Width 14,,;
    ColorNormal "BtnText/BtnFace",;
    Text "&Copy",;
    OnClick CLASS::PUSHBUTTON3_ONCLICK,,;
    Height 1.5,,;
    Left 33.5
DEFINE PUSHBUTTON PUSHBUTTON4 OF THIS;
PROPERTY;
    Group .T.,;
    Top 10,,;
    PageNo 1,,;
    Width 14,,;
    ColorNormal "BtnText/BtnFace",;
    Text "&Paste",;
    OnClick CLASS::PUSHBUTTON4_ONCLICK,,;
    Height 1.5,,;
    Enabled .F.,;
    Left 33.5
DEFINE ENTRYFIELD ENTRYFIELD2 OF THIS;
PROPERTY;
    OnGotFocus CLASS::ENTRYFIELD2_ONGOTFOCUS,,;
    Value "",;
    Border .T.,;
    Top 6,,;
    PageNo 1,,;
    Width 25,,;
    ColorNormal "WINDOWTEXT/WINDOW",;
    Height 1.5,,;
    ColorHighLight "WINDOWTEXT/WINDOW",;
    Left 3
DEFINE TEXT TEXT1 OF THIS;
PROPERTY;
    Border .F.,;
    Top 2,,;
    PageNo 1,,;
    Width 20.166,,;
    ColorNormal "BTNTEXT/BTNFACE",;
    Text "C&opy/Cut from here:",;
    Height 0.7646,,;
    Left 3.5
DEFINE TEXT TEXT2 OF THIS;
```

```

PROPERTY;
    Border .F.,;
    Top 5,,;
    PageNo 1,,;
    Width 18.166,,;
    ColorNormal "BTNTEXT/BTNFACE",;
    Text "Paste to here:",;
    Height 0.7646,,;
    Left 3.5
Procedure PUSHBUTTON1_OnClick
    Form.EntryLast.Undo()
    This.Enabled = .F.
Return
Procedure PUSHBUTTON2_OnClick
    IF .NOT. ISBLANK(TRIM(Form.EntryLast.Value))
        Form.EntryLast.Cut()
        Form.Pushbutton1.Enabled = .T.
    ENDIF
RETURN
Procedure PUSHBUTTON3_OnClick
    IF .NOT. ISBLANK(TRIM(Form.EntryLast.Value))
        Form.EntryLast.Copy()
    ENDIF
RETURN
Procedure PUSHBUTTON4_OnClick
    Form.EntryLast.Paste()
    Form.Pushbutton1.Enabled = .T.
RETURN
Procedure ENTRYFIELD1_OnGotFocus
    Form.EntryLast = This
    Form.Pushbutton1.Enabled = .F.
    Form.Pushbutton2.Enabled = .T.
    Form.Pushbutton3.Enabled = .T.
    Form.Pushbutton4.Enabled = .F.
Return
Procedure ENTRYFIELD2_OnGotFocus
    Form.EntryLast = This
    Form.Pushbutton1.Enabled = .F.
    Form.Pushbutton2.Enabled = .F.
    Form.Pushbutton3.Enabled = .F.
    Form.Pushbutton4.Enabled = .T.
RETURN
Procedure Form_OnOpen
    This.EntryField1.SetFocus()
Return
ENDCLASS

```

See Also

Cut(), EditCopyMenu, Paste(), Undo()

Count()

Returns the number of prompts in a list box or the number of elements in an associated array.

Property of class

ARRAY, LISTBOX

Description

Use Count() when you can't anticipate the number of prompts a list box might have at run time. For example, when you specify FILE *.* for the DataSource property, the number of prompts varies when files are added to or deleted from the default directory.

You can use Count() to control loops that evaluate user choices in a multiple-choice list box. For example, you can see which prompts were chosen by evaluating each prompt with the Selected() method in a DO...WHILE loop.

Make a list box multiple-choice by setting the Multiple property to true (.T.).

You can also use Count() to determine the number of elements in an associated array.

Example

The following example defines a form that contains a listbox that displays names from the Animals.DBF table. Property Multiple .T. lets the user select more than one listbox prompt. The OnRightMouseDown property calls procedure Checked, which uses the methods Count() and Selected() to send the selected prompts to the Command window results pane with each OnRightMouseDown:

```
LOCAL f
f = NEW XFORM()
f.Open()

CLASS XFORM OF FORM
  this.Left = 52.80
  this.Width = 40.60
  this.Text = "Animals of the World"
  this.HelpId = ""
  this.OnRightMouseDown = CHECKED
  this.HelpFile = ""
  this.Top = 3.12
  this.Height = 20.00

  DEFINE LISTBOX LB1 OF THIS;
  PROPERTY;
    Left 10.00;;
    ColorNormal "N/W*";
    Width 20.00;;
    DataSource "FIELD ANIMALS->NAME";
    Multiple .T.;;
    ColorHighLight "W+/B";
    Top 4.00;;
    Height 12.00
ENDCLASS
```

```

PROCEDURE Checked
FOR i=1 TO Form.LB1.Count()
    ? Form.LB1.Selected(i)
NEXT i
RETURN

```

See Also

FOR...NEXT, LISTCOUNT(), LISTSELECTED(), Selected(), Multiple

CUATab

Determines cursor behavior when you press Tab while on a Browse or Editor object.

Property of class

BROWSE, EDITOR

Data type Logical

Default The default for CUATab is true (.T.).

Description

When CUATab is .T. (the default value), pressing Tab moves to the next control in the Form's tab order. When CUATab is .F., pressing Tab moves to the next field in a Browse object or moves the cursor to the next tab stop position in an Editor object.

Example

```

local f
f = new SAMPLEFORM()
f.Open()
CLASS SAMPLEFORM OF FORM
    this.Top = 2
    this.Width = 76
    this.Text = "Sample Form"
    this.View = "animals.dbf"
    this.Height = 20
    this.Left = 11
    DEFINE BROWSE BROWSE1 OF THIS;
        PROPERTY;
            Top 3;;
            Width 56;;
            Alias "ANIMALS",;
            Height 16;;
            CUATab .F.,;
            Left 1
    DEFINE OKBUTTON OKBUTTON1 OF THIS;
        PROPERTY;
            Group .T.,;
            Top 3;;
            Width 14;;
            OnClick CLASS::OKBUTTON1_ONCLICK,;

```

```

        Height 1.5,;
        Left 60
    Procedure OKBUTTON1_OnClick
        Form.Close()
    Return
ENDCLASS

```

See Also

_curobj, Before, NextObj, SET CUAENTER

CurSel

Determines which prompt is selected in a list box or tab box.

Property of class

LISTBOX, TABBOX

Data type

Numeric

Default

The default for CurSel is 0 for a list box, 1 for a tab box.

Description

Use CurSel to specify which list box or tab box prompt is highlighted.

CurSel lets you place the highlight on a prompt other than the prompt last chosen by the user. For example, an OnLostFocus subroutine might use CurSel to highlight the top prompt each time the user moves focus to a different object.

Example

NEW operator syntax:

```

this.Lbl1 = NEW LISTBOX(this)
this.Lbl1.Top=4
this.Lbl1.Left=6
this.Lbl1.Width=20
this.Lbl1.Height=12
this.Lbl1.DataSource="FIELD Animals->Name"
this.Lbl1.CurSel= 3

```

DEFINE object syntax:

```

DEFINE LISTBOX Lbl1 OF THIS;
PROPERTY;
Top 4, Left 6, Width 20, Height 12,;
DataSource "FIELD Animals->Name",;
CurSel 3

```

See Also

Count(), Selected()

Cut()

D

Cuts selected text and places it on the Windows clipboard.

Property of class

BROWSE, COMBOBOX, EDITOR, ENTRYFIELD, SPINBOX

Description

Use Cut() when the user has selected text and wants to remove it from the edit control and place it on the Windows clipboard. The action of Cut() is identical to the Cut menu item on the standard Windows Edit menu.

If you have assigned a menubar to the form, you can use the menubar's EditCutMenu property instead of using the Cut() property of individual objects on the form. For more information, see EditCutMenu.

Example

See Copy() for an example.

See Also

Copy(), EditCutMenu, Paste(), Undo()

DataLink

Links an object in a form to a table field.

Property of class

CHECKBOX, COMBOBOX, EDITOR, ENTRYFIELD, OLE, RADIOBUTTON, SCROLLBAR, SPINBOX

Data type

Character

Default

The default for DataLink is an empty string.

Description

If you create a check box, entry field, radio button, or spin box to access table data, you need to establish a link between the object and a field in the table with DataLink. Once you've established the link, any changes entered through the object are inserted into the field automatically.

When the user moves from record to record, the change is reflected in the object. For example, if an entry field is linked to a table field and the user moves to another record while the form is open, the value in the entry field is updated with the current table field value. If the form is closed when the user moves to another record, the value in the entry field is not updated.

The DataLink property is similar to the DataSource property. However, data displayed through the DataLink property can be changed, while data displayed through the DataSource property is always read-only. For example, the DataSource property of a combo box determines its prompts (which are read-only values), while the DataLink property determines which field is affected by the user's input.

Note You can select a field with the Choose Field dialog box. To access the Choose Field dialog box, click the Tool button next to the DataLink item in the Inspector.

Example

NEW operator syntax:

```
Company = NEW ENTRYFIELD(this)
Company.Top = 3
Company.Left = 1
Company.Datalink = "Company"
Company.ColorHighlight = "B/W"
Company.ColorNormal = "W/B"
Company.Function = "@!"
```

DEFINE object syntax:

```
DEFINE Entryfield Company OF THIS;
  AT 3,1;
  Property;
    Datalink "Company",;
    ColorHighlight "B/W",;
    ColorNormal "W/B",;
    Function "@!"
```

See Also

Alias, DataSource, DEFINE, Fields, Refresh()

DataSource

Determines which data is displayed in a list box, a combo box, or an image object.

Property of class

COMBOBOX, IMAGE, LISTBOX, TABBOX

Data type

Character

Default

The default for DataSource is an empty string.

Description

You need to specify a valid value for the DataSource property to display an image in an image object, or prompts in a list box or a combo box.

The DataSource property is similar to the DataLink property. However, data displayed through the DataLink property can be changed, while data displayed through the DataSource property is always read-only.

You can specify one of five DataSource values for a list box or a combo box:

- 1 FILE *<filename skeleton expC>* creates prompts from file names in the current default directory.
- 2 FIELD *<field name>* creates prompts from all the values in a field in a table file. Each prompt represents a record, and you can move from record to record by selecting different prompts. To create prompts that don't move from record to record, copy the field into an array object with COPY TO ARRAY, then use the DataSource ARRAY *<array name>* option (described in this list).
- 3 STRUCTURE creates prompts from all the field names in a table.
- 4 ARRAY *<array name>* creates prompts from elements in an array object.
- 5 TABLES creates prompts from the names of all tables in the currently open database. See OPEN DATABASE for information on databases.

To evaluate which prompts were chosen from a list box, use LISTSELECTED() or Selected().

Note You can specify one of three DataSource values for an image object:

- RESOURCE *<resource id><DLL name>* designates a resource within a DLL file. *<resource id>* is a numeric literal that identifies a bitmap image in the DLL file. *<DLL name>* is the name of the DLL file and must include the file name extension if the DLL file isn't already in memory.
- FILENAME *<filename>* is the name of a file containing a bitmap image.
- BINARY *<binary field>* is the name of a binary field containing bitmap images.

Example

NEW operator syntax:

```
this.COMBOBOX1 = NEW COMBOBOX(this)
this.COMBOBOX1.DataLink = "Company"
this.COMBOBOX1.Value = "General Consolidated"
    DataSource = "FIELD COMPANY"
* or
*   DataSource = "STRUCTURE"
* or
*   DataSource = "FILE '*.PRG'"
```

D

Default

```
* or
* DataSource = "TABLES"
* or
* DataSource = "Array 'ComboList'"
```

DEFINE object syntax:

```
DEFINE COMBOBOX COMBOBOX1 OF THIS;
PROPERTY;
    DataLink "Company",;
    Value "General Consolidated",;
    DataSource "FIELD COMPANY"
* or
* DataSource "STRUCTURE"
* or
* DataSource "FILE '*.PRG'"
* or
* DataSource "TABLES"
* or
* DataSource "Array 'ComboList'"
```

See Also

DEFINE

Default

Determines if a pushbutton is a default pushbutton.

Property of class

PUSHBUTTON

Data type

Logical

Default

The default for Default is false (.F.).

Description

Use the Default property to make a pushbutton the default pushbutton when the user submits a form by pressing *Enter*. Setting the Default property of a pushbutton to true gives the pushbutton a visual highlight that identifies it as the default.

Setting the Default property to true (.T.) causes two things to happen when the user presses *Enter*:

- The OnClick subroutine of the default pushbutton executes.
- The ID property of the default pushbutton is passed to the OnSelection routine of the parent form.

The OnSelection subroutine can use the ID value to make branching decisions.

However, if the user clicks on any pushbutton, the OnClick subroutine of that pushbutton executes. The ID value of that pushbutton is passed to the OnSelection subroutine, even if the Default property of another pushbutton is true.

If you give more than one pushbutton a Default value of true, the last pushbutton to get the value is the default. If you give all pushbuttons a Default value of false, the pushbutton that you created first is the default.

D

Example

NEW operator syntax:

```
BUTTON1 = New PUSHBUTTON(this)
BUTTON1.Text = "BUTTON1"
BUTTON1.Top = 3.00
BUTTON1.Onclick = Push1
BUTTON2 = New PUSHBUTTON(this)
BUTTON2.Text = "BUTTON2"
BUTTON2.Top = 8.00
BUTTON2.Onclick = Push2
BUTTON2.Default = .T.
```

DEFINE object syntax:

```
DEFINE PUSHBUTTON BUTTON1 OF THIS;
PROPERTY;
  Text "BUTTON1",;
  Top 3.00,;
  Onclick Push1
DEFINE PUSHBUTTON BUTTON2 OF THIS;
PROPERTY;
  Text "BUTTON2",;
  Top 8.00,;
  Onclick Push2,;
  Default .T.
```

See Also

DEFINE, OnClick

Delete

Determines if records in a browse object can be marked for deletion.

Property of class

BROWSE

Data type

Logical

Delete()

Default

The default for Delete is true (.T.).

Description

Set Delete to false to prevent users from marking records for deletion. For example, an application might allow certain users to view records but, for reasons of security, not allow them to delete records.

Note A record is not removed from a table when it is marked for deletion; it is removed only when you execute the PACK command or when the user chooses the Utilities | Pack command of the Table menu.

Example

NEW operator syntax:

```
DEFINE BROWSE Br1 OF this
  this.Br1.Delete=.F.
  this.Br1.Top=4
  this.Br1.Left=3
  this.Br1.Width=32
  this.Br1.Height=12
```

DEFINE object syntax:

```
DEFINE BROWSE Br1 OF THIS FROM 4,3 TO 16,35;
  PROPERTY;
  Delete .F.
```

See Also

Append, Modify

Delete()

Deletes an element from a one-dimensional array object or a row or column from a two-dimensional array object.

Property of class

ARRAY

Description

Use Delete() to delete selected elements from an array without changing the size of the array. Delete() does the following:

- Deletes an element from a one-dimensional array, or deletes a row or column from a two-dimensional array
- Moves all remaining elements toward the beginning of the array (up if a row is deleted, to the left if an element or column is deleted)
- Inserts .F. values in the last position or positions

For information about deleting elements and moving remaining elements toward the *end* of the array, see `Insert()`. For information about replacing elements without moving remaining elements, see `Fill()`. To change the size of an array, use `Grow()` or `Resize()`.

Example

```
USE Animals.DBF
* Initialize an array object
ObjArr=NEW ARRAY(RECCOUNT(),3)
* Fill array with values from Animals.DBF
COPY TO ARRAY ObjArr FIELDS Name, Area, Weight
* Use DELETE() to remove the Weight values from
* column 3 and replace with false (.F.)
ObjArr.DELETE(3,2)
* Display results of DELETE()
FOR i=1 TO RECCOUNT()
? ObjArr[i,1],ObjArr[i,2],ObjArr[i,3]
NEXT i
```

See Also

`ADEL()`, `Add()`, `Insert()`, `RemoveKey()`

DesignView

Designates a view (.QBE or table) that is used when designing a form.

Property of class

FORM

Data type

Character

Default

The default for DesignView is an empty string.

Description

Use DesignView to facilitate creating and datalinking a form when you don't want to assign a View property to the form. The value in DesignView is ignored at runtime.

There are two main instances in which you may want to use DesignView instead of View.

- If you know which tables will be open when the form is opened at runtime, use DesignView to avoid opening the tables again when the form is opened. For example, if you are designing a Search dialog box that will be opened only when a specific table is already open, set the DesignView property of the dialog box instead of the View property.

- If you don't know which tables will be open when the form is opened at runtime, but need certain tables open to design the form, use DesignView to avoid specifying at design time which tables will be open at runtime.

If you specify a View property for a form, you should not also specify a DesignView property. If you want to design multiple forms having different DesignView properties, you should design the forms in different sessions.

See Also

Alias, DataLink, DataSource, View

Dimensions

Specifies the number of dimensions in an array object.

Property of class

ARRAY

Data type

Numeric

Description

Use Dimensions to find out whether an array object is one-dimensional or multi-dimensional.

Dimensions is a read-only property.

Example

```
USE Customer.DBF
* Initialize an array object
ObjArr=NEW ARRAY(RECCOUNT())
* Fill 1-dimensional array with values
* from Name field of Customer.DBF
GO TOP
FOR i=1 TO RECCOUNT()
    ObjArr[i] = Customer->Name
    SKIP
NEXT i
* Use Dimensions to return array dimensions
* and GROW() to add a second column to the
* existing array and enter Ytd_Sales data
* in new second column.
IF ObjArr.Dimensions=1
    ObjArr.GROW(2)
    GO TOP
    FOR i=1 TO RECCOUNT()
        ObjArr[i,2]=Customer->Ytd_Sales
        SKIP
```

```
    NEXT i
ENDIF
* Display Contents of 2-dimensional array
FOR i=1 TO RECCOUNT()
? ObjArr[i,1], ObjArr[i,2]
NEXT i
```

See Also
Add(), ALen(), Delete(), Fields(), Insert(), Resize()

Dir()

Stores the name, size, date stamp, time stamp, and DOS attributes of files to an array object.

Property of class
ARRAY

Description
Use the Dir() method to store file information to an array object. The Dir() method is similar to the ADIR() function.

Dir() accepts two optional parameters:

- *<filename skeleton expC>* identifies file names with a wildcard character string. For example, to return information on tables only, use "*.DBF" as *<filename skeleton expC>*. If you omit *<filename skeleton expC>*, Dir() stores information about all files except hidden or system files in the current directory.
- *<file attribute list expC>* specifies one or more DOS file attributes. The meaning of each attribute is as follows:

Character	Meaning
D	Include directories
H	Include hidden files
S	Include system files
V	Include volume label only

When you specify more than one letter for *<file attribute list expC>*, include all the letters between quotation marks. If you specify a value for *<file attribute list expC>*, also include a value for *<filename skeleton expC>*.

When you include the letter V in *<file attribute list expC>*, Dir() ignores *<filename skeleton expC>* and all other characters in the attribute list and stores the volume label to a one-element array object.

The array object is dynamically sized to fit all returned information.

Example

The following example uses Array() to initialize an array object and Dir() to fill the array with DOS directory file information. The FOR...NEXT loop traverses the array to display the results of the Dir() operation. Note the use of Size to return the number of elements in the array. Dividing the number of elements by the number of file attributes that DIR() copies yields the number of rows in the array:

```
ObjArr=NEW ARRAY()
ObjArr.DIR("*.PRG","D")
FOR i=1 TO ObjArr.Size/5
? ObjArr[i,1],ObjArr[i,2] AT 20,;
  ObjArr[i,3] AT 35,ObjArr[i,4] AT 45,;
  ObjArr[i,5] AT 55
NEXT i
```

See Also

ADIR(), Fields()

DirExt()

Stores the name, size, date stamp, time stamp, and DOS and Windows 95 attributes of files to an array object.

Property of class

ARRAY

Description

DirExt() is identical to Dir(), but with extra columns for Windows 95 file information. For more information, see Dir().

Example

```
DirList = NEW ARRAY(1)
? DirList.DirExt()
```

See Also

Dir()

DisabledBitmap

Specifies the graphic image to display in a pushbutton when the pushbutton is disabled.

Property of class

PUSHBUTTON

D

Data type

Character

Default

The default for DisabledBitmap is an empty string.

Description

Use DisabledBitmap to indicate visually when a pushbutton is not available for use.

A pushbutton is disabled when

- Its Enabled property is set to false (.F.).
- Its When property returns a value of false.

The DisabledBitmap setting can take one of two forms:

- 1 RESOURCE <resource id> <dll name> specifies a bitmap resource and the DLL file that holds it. (A DLL file is a precompiled library of external routines and resources written in non-dBASE languages such as C and Pascal.) You can obtain the resource ID with a resource editor such as Resource Workshop.
- 2 FILENAME <filename> specifies a bitmap file (which usually has the file-name extension .BMP).

If you specify a character string with Text and an image with DisabledBitmap, the image is displayed with the character string when the pushbutton is disabled. When you specify no image with DisabledBitmap, the Text property character string is displayed with low intensity when the pushbutton is disabled.

Note You can select a bitmap file with the Choose Bitmap dialog box, which you can access by clicking the Tool button next to the DisabledBitmap item in the Inspector.

Example

NEW operator syntax:

```
Go = NEW PUSHBUTTON(this)
Go.DisabledBitmap = "RESOURCE 20 EntryIcn.dll"
* or
* Go.DisabledBitmap = "FILENAME PushDis.bmp"
* or
* Go.DisabledBitmap = "BINARY PushDis"
```

DoVerb()

DEFINE object syntax:

```
DEFINE PUSHBUTTON Go OF THIS;  
  Property;  
    DisableBitmap "RESOURCE 20 EntryIcn.dll"  
  * or  
  * DisableBitmap "FILENAME PushDis.bmp"  
  * or  
  * DisableBitmap "BINARY PushDis"
```

See Also

DEFINE, DownBitmap, Enabled, FocusBitmap, UpBitmap

DoVerb()

Starts an action in an OLE server application.

Property of class

OLE

Description

Use DoVerb() to initiate an action from an OLE document stored in an OLE field and to specify what action to take.

Every OLE object accepts one or more verbs. Each verb determines which actions are taken, and each is represented by a number. For example, most sound applications accept one of two verbs:

- 0 (Play) plays a sound stored in an OLE field.
- 1 (Edit) opens the Sound Recorder to edit the sound.

Some OLE documents have only one verb. For example, Quattro Pro for Windows has only 0 (The Edit verb).

DoVerb accepts two parameters:

- *<OLE verb>* The numeric value of the verb.
- *<title>* A text string to display in the title bar of the server window (if *<OLE verb>* causes a window to appear).

Note You can determine which verbs are accepted by an OLE document with the following steps:

- 1 In a browse object, double-click on the OLE field containing the document. This opens the OLE viewer.
- 2 Open the Edit menu. The OLE object is the last menu item; for example, if the OLE was created by the Windows Sound Recorder, the last menu item is Sound Object.

- 3 Click once on this menu item; if the OLE document accepts more than one verb, they are displayed as menu items in a child menu. If the OLE document accepts only one verb, that verb is executed.

You can also view OLE verbs through the Windows registration database, which you can examine with the Registration Info Editor. You access the editor by starting a session in Windows in *verbose mode* and specifying *regedit* as a parameter, as with:

```
win regedit /v
```

D

Example

The following program creates a form that displays an OLE object and a pushbutton. The OnClick subroutine of the pushbutton uses DoVerb to start an action in an OLE server.

```
LOCAL f
f = NEW XFORM()
f.Open()

CLASS XFORM OF FORM
  this.Left = 22.40
  this.Width = 103.60
  this.MousePointer = 1
  this.Text = "Form"
  this.HelpId = ""
  this.HelpFile = ""
  this.View = "PICTURES.QBE"
  this.Top = 1.06
  this.Height = 24.53

  DEFINE OLE OLE1 OF THIS;
  PROPERTY;
  Left 2.00;;
  Width 100.00;;
  DataLink "PICTURES->BITMAPOLE",;
  Border .T.,;
  Top 1.00;;
  Height 20.00

  DEFINE PUSHBUTTON BUTTON1 OF THIS;
  PROPERTY;
  Left 39.00;;
  ColorNormal "N/W",;
  Width 22.00;;
  OnClick CLASS::BUTTON1_ONCLICK,;
  Text "Click me to edit this image.",;
  Default .T.,;
  Top 22.00;;
  Height 2.00

  Procedure BUTTON1_OnClick
  form.OLE1.DoVerb(0, "This Title Is Optional")

ENDCLASS
```

See Also
CLASS OLE

DownBitmap

Specifies the graphic image to display in a pushbutton when the user presses the button.

Property of class
PUSHBUTTON

Data type
Character

Default
The default for DownBitmap is an empty string.

Description
Use DownBitmap to give visual confirmation when the user clicks a pushbutton. When the user releases the mouse button or moves the pointer off the pushbutton, the image or text specified by UpBitmap or Text is displayed.

The DownBitmap setting can take one of two forms:

- 1 RESOURCE *<resource id>* *<dll name>* Specifies a bitmap resource and the DLL file that holds it. (A DLL file is a precompiled library of external routines and resources written in non-dBASE languages such as C and Pascal.) You can obtain the resource ID with a resource editor such as Resource Workshop.
- 2 FILENAME *<filename>* specifies a bitmap file (which usually has the file-name extension .BMP).

When you specify a character string for the pushbutton with Text and an image with DownBitmap, the image is displayed with the character string when the user clicks the pushbutton.

Note You can select a bitmap file with the Choose Bitmap dialog box. To access the Choose Bitmap dialog box, click the Tool button next to the DownBitmap item in the Inspector.

Example
See UpBitmap for an example of DownBitmap.

See Also
DEFINE, DisabledBitmap, Enabled, FocusBitmap, Text, UpBitmap

DropDownHeight

Specifies the number of lines displayed in the list portion of the combo box.

Property of class

COMBOBOX

D

Data type

Numeric

Default

The default for DropDownHeight is 6.

Description

Use DropDownHeight to specify how much information will appear when a user drops down a list from a combo box.

Example

In the following example, the dropdown portion of the list contains either 10 lines or the total number of list items available, whichever is smaller.

```

cbararray=NEW ARRAY(5)           && Create array containing 5 elements
cbararray2=NEW ARRAY(15)        && Create array containing 15 elements

f = NEW FORM()
c1 = NEW COMBOBOX(f)
c1.DataSource = 'ARRAY cbararray' && Acceptable values come from array
c1.Style = 2                     && Only array items can be selected
c1.DropDownHeight = IIF(cbararray.size<10,cbararray.size,10) && 5 lines
c2 = NEW COMBOBOX(f)
c2.Left = c1.Left+20
c2.DataSource = 'ARRAY cbararray2'
c2.Style = 2
c2.DropDownHeight = IIF(cbararray2.size<10,cbararray2.size,10) && 10 lines
f.Open()

```

See Also

Style

EditCopyMenu

Specifies a menu item that copies selected text from a control to the Windows clipboard.

Property of Class

MENUBAR

Data type

Object reference

Description

EditCopyMenu contains a reference to a menu object users select when they want to copy text.

You can use the EditCopyMenu property of a form's menubar to copy selected text to the Windows clipboard from any edit control in the form, instead of using the control's Copy() property. In effect, EditCopyMenu calls Copy() for the active control. This lets you provide a way to copy text with less programming than would otherwise be needed. The Copy menu item is automatically disabled (greyed out) when no text is selected, and enabled when text is selected.

For example, suppose you have a Browse object (b) and an Editor object (e) on a form (f). To implement text copying, you could specify actions that would call b.Copy() or e.Copy() whenever a user wanted to copy text. However, if you use a menubar, you can set the EditCopyMenu property to the menu item the user will select to copy text. Then, when the user selects that menu item, the text is automatically copied to the Windows clipboard from the currently active control. You don't need to use the control's Copy() property at all.

If you use the Menu Designer to create a menubar, EditCopyMenu is automatically set to an item named Copy on a pulldown menu named Edit when you add the Edit menu to the menubar:

```
this.EditCopyMenu = this.Edit.Copy
```

Example

See WindowMenu for an example.

See Also

CLASS MENUBAR, Copy(), EditCutMenu, EditPasteMenu, EditUndoMenu, WindowMenu

EditCutMenu

Specifies a menu item that cuts selected text from a control and places it on the Windows clipboard.

Property of Class

MENUBAR

Data type

Object reference

Description

EditCutMenu contains a reference to a menu object users select when they want to cut text.

You can use the EditCutMenu property of a form's menubar to cut (delete) selected text and place it on the Windows clipboard from any edit control in the form, instead of using the control's Cut() property. In effect, EditCutMenu calls Cut() for the active control. This lets you provide a way to copy text with less programming than would otherwise be needed. The Cut menu item is automatically disabled (greyed out) when no text is selected, and enabled when text is selected.

For more information, see EditCopyMenu.

Example

For an example of EditCutMenu, see WindowMenu.

See Also

CLASS MENUBAR, Cut(), EditCopyMenu, EditPasteMenu, EditUndoMenu, WindowMenu

EditPasteMenu

Specifies a menu item that copies text from the Windows clipboard to the currently active edit control.

Property of Class

MENUBAR

Data type

Object reference

Description

EditPasteMenu contains a reference to a menu object users select when they want to paste text to the cursor position in the currently active edit control.

E

You can use the `EditPasteMenu` property of a form's menubar to paste text from the Windows clipboard into any edit control in the form, instead of using the control's `Paste()` property. In effect, `EditPasteMenu` calls `Paste()` for the active control. This lets you provide a way to paste text with less programming than would otherwise be needed. The Paste menu item is automatically disabled (greyed out) when the clipboard is empty, and enabled when text is copied or cut to the clipboard.

For more information, see `EditCopyMenu`.

Example

For an example of `EditPasteMenu`, see `WindowMenu`.

See Also

`CLASS MENUBAR`, `Paste()`, `EditCopyMenu`, `EditCutMenu`, `EditUndoMenu`, `WindowMenu`

EditUndoMenu

Specifies a menu item that reverses the effects of the last Cut, Copy, or Paste action.

Property of Class

`MENUBAR`

Data type

Object reference

Description

`EditUndoMenu` contains a reference to a menu object users select when they want to undo their last Cut, Copy, or Paste action.

You can use the `EditUndoMenu` property of a form's menubar to undo a Cut or Paste action from any edit control in the form, instead of using the control's `Undo()` property. In effect, `EditUndoMenu` calls `Undo()` for the active control. This lets you provide a way to undo with less programming than would otherwise be needed.

For more information, see `EditCopyMenu`.

Example

For an example of `EditUndoMenu`, see `WindowMenu`.

See Also

`CLASS MENUBAR`, `Undo()`, `EditCopyMenu`, `EditCutMenu`, `EditPasteMenu`, `WindowMenu`

Element()

Returns the number of a specified element in an array object.

Property of class

ARRAY

Description

Use Element() when you know the subscripts of an element in a two-dimensional array object and need the element number for use with another method, such as Scan(). The Element() method is similar to the AELEMENT() function.

An element number indicates the sequential position of an element in the array. In one-dimensional arrays, the element number is the same as the subscript, so there is no need to use Element(); for example, Element(3) returns 3, Element(5) returns 5, and so on.

Element() accepts two parameters:

- *<subscript1 expN>* is the first subscript number of the element. In a one-dimensional array, *<subscript1 expN>* is the same as the element number. In a two-dimensional array, *<subscript1 expN>* identifies a row.
- *<subscript2 expN>* specifies the column of the element when the array object is two-dimensional. If you omit *<subscript2 expN>*, dBASE assumes the value 1. If the array object is one-dimensional, dBASE ignores *<subscript2 expN>*.

Element() is the inverse of Subscript(), which returns the row or column subscript number of an element when you specify the element number.

Example

```
USE Animals.DBF
* Initialize an array object
ObjArr=NEW ARRAY(RECCOUNT(),3)
* Fill array with values from Animals.DBF
COPY TO ARRAY ObjArr FIELDS Name, Area, Weight
* Use ELEMENT() to return the array element number
* of all Name values in the 1st column.
FOR i=1 TO RECCOUNT()
? ObjArr[i,1], ObjArr.Element(i,1)
NEXT i
```

See Also

AELEMENT(), Scan(), Subscript()

E

Enabled

Determines if an object can be selected.

Property of class

BROWSE, CHECKBOX, COMBOBOX, EDITOR, ENTRYFIELD, FORM, LISTBOX, MENU, OLE, PAINTBOX, PUSHBUTTON, RADIOBUTTON, SCROLLBAR, SPINBOX, TABBOX

Data type

Logical

Default

The default for Enabled is true (.T.).

Description

When you set the Enabled property of an object to true, the user can select and use the object. When you set the Enabled property to false (.F.), the object is dimmed and the user can't select or use the object.

You usually set the Enabled property to false when a required condition is not met. For example, the Valid subroutine of an entry field might disable an OK pushbutton when the user hasn't entered correct data into the entry field.

Example

NEW operator syntax:

```
Compcode = NEW ENTRYFIELD(this)
Compcode.Top = 3
Compcode.Left = 1
Compcode.Datalink = "CompCode"
Compcode.Enabled = .F.
* user cannot change Compcode
```

DEFINE object syntax:

```
DEFINE Entryfield Compcode OF THIS;
  AT 3,1;
  Property;
  Datalink "CompCode",;
  Enabled .F.
```

See Also

Visible

EscExit

Determines if the user can close a form by pressing *Esc*.

Property of class

FORM

Data type

Logical

Default

The default for EscExit is true (.T.).

Description

Set EscExit to false (.F.) to prevent the user from closing a form by pressing *Esc*.

Example

NEW operator syntax:

```
LOCAL f
f=NEW EntryForm()
f.OPEN()
CLASS EntryForm OF FORM
  this.Top=2
  this.Left=2
  this.Height=10
  this.Width = 20
  this.EscExit=.F.
```

DEFINE object syntax:

```
DEFINE FORM EntryForm ;
  PROPERTY;
  Top 2, Left 2, Height 10, Width 20,;
  EscExit .F.
```

See Also

ON ESCAPE, SET ESCAPE

E

Execute()

Sends a command string to a DDE server application in its own language.

Property of class

DDELINK

Description

Use the Execute() method to send macro instructions to a server application.

Execute() requires the *<command>* parameter, which is a string consisting of at least one command in the language of the server. Surround each command with the delimiters required by the server application. For example, each Quattro Pro command is surrounded with braces ({}). Some applications accept multiple commands separated by brackets. For information, consult your DDE server documentation.

Before you can send a command string to a server, you must open the server application, open the document, and establish a DDE link. For information on establishing DDE links, see Initiate() and Chapter 26 in the *Programmer's Guide*.

Example

```
PUBLIC LinkObj
LinkObj = NEW DDELINK()
LinkObj.Initiate("QPW", "Demo.WB1")
? LinkObj.Execute('{||OPEN "C:INFO\Data.TXT",W}');
    && Creates text file Data.TXT
? LinkObj.Execute('{||WRITELN +A1}');
    && Send contents of QPW cell A1 to;
    Data.TXT
? LinkObj.Execute('{||WRITELN +A2}');
    && Send contents of QPW cell A2 to;
    Data.TXT
? LinkObj.Execute('{||CLOSE}');
    && Closes Data.TXT
```

See Also

Advise(), Initiate(), OnNewValue, Peek(), Poke(), Server, Terminate(), Timeout, Topic, Unadvise()

Fields

Specifies the fields to display in a browse object, and the field options to apply to each field.

Property of class

BROWSE

Data type

Character

F

Default

The default for Fields is an empty string.

Description

The value you place in Fields must be surrounded by quotation marks and should have the following format:

```
"<field 1> [<field option list 1>] |
<calculated field 1> = <exp1> [<calculated field option list 1>]
[, <field 2> [<field option list 2>] |
<calculated field 2> = <exp1> [<calculated field option list 2>], ...]"
```

The fields are displayed in the order they're listed, and the field option lists affect the way the fields are displayed. The options are as follows:

/<column width>	The width of the column within which <field 1> appears when <field 1> is character type
/B = <exp 1>, <exp 2> [/F]	RANGE option; forces any value entered in <field 1> to fall within <exp 1> and <exp 2>, inclusive. RANGE REQUIRED option; the /F option prevents a previously entered value from being accepted if it doesn't fall between <exp 1> and <exp 2>, inclusive
/H = <expC>	HEADER option; causes <expC> to appear above the field column in the Table Records window, replacing the field name
/P = <expC>	PICTURE option; displays <field 1> according to the PICTURE or FUNCTION clause <expC>
/R	READ-ONLY option; specifies that <field 1> is read-only and can't be edited
/V = <condition> [/F] [/E = <expC>]	VALID option; allows a new <field 1> value to be entered only when <condition> evaluates to .T. VALID REQUIRED option; the /F option prevents the user from leaving <field 1> and the editing session from ending until <condition> evaluates to .T.
	ERROR MESSAGE option; /E = <expC> causes <expC> to appear when <condition> evaluates to .F.
/W = <condition>	WHEN option; allows <field 1> to be edited only when <condition> evaluates to .T.

Read-only calculated fields are composed of an assigned field name and an expression that results in the calculated field value, as with *Commission = Rate * Saleprice*. Options for calculated fields affect the way these fields are displayed. These options are as follows:

<code>/<column width></code>	The width of the column within which <i><calculated field T></i> is displayed
<code>/H = <expC></code>	Causes <i><expC></i> to appear above the calculated field column in the Table Records window, replacing the calculated field name

Calculated fields are read-only.

Note You can select a field with the Browse Field Picker. To access the Browse Field Picker, click the Tool button next to the Fields item in the Inspector.

Example

NEW operator syntax:

```
this.Br1 = NEW BROWSE(this)
this.Br1.Alias="Contact"
this.Br1.Fields="CompCode /R,Contact"
this.Br1.Top=4
this.Br1.Left=3
this.Br1.Width=32
this.Br1.Height=12
```

DEFINE object syntax:

```
DEFINE BROWSE Br1 OF THIS;
PROPERTY;
Top 4, Left 3, Width 32, Height 12;;
Alias "Contact",;
Fields "CompCode /R, Contact"
```

See Also

Alias, DataLink, DataSource, SET FIELDS

Fields()

Stores the structure information of the current table to an array object and returns the number of fields whose characteristics are stored.

Property of class

ARRAY

Description

Use the Fields() method to store information about the structure of the current table in an array object. Each row in the array object contains information on a single field in the current table. The Fields() method is similar to the AFIELDS() function.

Fields() dynamically sizes the array object so the number of rows is at least equal to the number of fields in the current table, and the number of columns is at least four. If the array contains more elements than are needed to store the table structure, the extra elements are left unchanged.

The following table shows which field characteristics Fields() stores, and in which column the information is placed:

Column 1	Column 2	Column 3	Column 4
Field name (character data type)	Field type (character data type)	Field length (numeric data type)	Decimal places (numeric data type)

F

dBASE uses the following codes for field types: C—character, D—date, L—logical, M—memo, N—numeric, F—float, G—ole, B—binary.

Fields() stores the same information into an array object that COPY TO...STRUCTURE EXTENDED stores into a table, except Fields() doesn't create a row containing FIELD_IDX information.

Example

```
USE Customer.DBF
* Initialize an array object to hold
* the structure of Customer.DBF.
ObjArr=NEW ARRAY(FLDCOUNT(),4)
* Fill array with .DBF file structure
* information.
ObjArr.Fields()
* Display array contents in the Command
* window results pane.
FOR i=1 TO FLDCOUNT()
? ObjArr[i,1],ObjArr[i,2] AT 20,;
  ObjArr[i,3] AT 25, ObjArr[i,4] AT 35
NEXT i
```

See Also

AFIELDS(), COPY TO...STRUCTURE EXTENDED, Dir(), Element(), Scan(), Sort(), Subscript()

FieldWidth

Specifies the width of a character field in a browse object.

Property of class
BROWSE

Data type
Numeric

Default

The default for FieldWidth is the length of the table field or the length of the field name, whichever is greater.

Description

Use FieldWidth to limit the display width of a character field in a browse object.

If the text in a character field exceeds the field length, the user accesses the entire field by scrolling within it.

Example

NEW operator syntax:

```
USE Animals.DBF
LOCAL f
f=NEW Brws()
f.OPEN()
CLASS Brws OF FORM
    this.Width = 45
    this.Br1=NEW BROWSE(this)
    this.Br1.Top=2
    this.Br1.Left=1
    this.Br1.Width=40
    this.Br1.Height=10
    this.Br1.FieldWidth=10
ENDCLASS
```

DEFINE object syntax:

```
* Form definition
DEFINE BROWSE Br1 OF THIS;
    PROPERTY;
    Top 2, Left 1, Width 40,,
    Height 10, FieldWidth 10
```

See Also

ShowHeading, ShowRecNo

Fill()

Inserts a specified value into one or more locations in an array object.

Property of class

ARRAY

Description

Use the Fill() method to insert a value into all or some elements of an array object. The Fill() method is similar to the AFILL() function.

Fill() accepts three parameters:

- *<exp>* is the value to insert into the array object.
- *<start expN>* is the element number at which to begin inserting *<exp>*. (If you know only the elements subscripts, you can use Element() to determine the element number.) If you do not specify *<start expN>*, dBASE begins at the first element in the array object.
- *<count expN>* is the number of elements in which to insert *<exp>*, starting at element *<start expN>*. If you do not specify *<count expN>*, Fill() inserts *<exp>* from *<start expN>* to the last element in the Array. If you specify a value for *<count expN>*, also specify a value for *<start expN>*.

If you do not specify *<start expN>* or *<count expN>*, Fill() fills all elements with *<exp>*.

Example

```
Use Animals.Dbf
* Initialize an array object
ObjArr=NEW ARRAY(RECCOUNT(),3)
* Fill all elements with the string "Test" and
* display contents in Command window results pane
ObjArr.Fill("Test")
FOR i=1 TO RECCOUNT()
? ObjArr[i,1],ObjArr[i,2], ObjArr[i,3]
NEXT i
```

See Also

AFILL(), Dir(), Element(), Fields(), Subscript()

First

Determines which object is the first to get focus when its parent form is opened.

Property of class

FORM

Data type

Object reference

Description

Use the First property to reference the object that gets focus initially when you open a form. First is a read-only property.

When a form holds two or more objects that users can select, the user can move from object to object by pressing *Tab* or *Shift+Tab*. The order in which focus moves from object to object is known as the *tabbing order*. The object reference variable contained in First references the object at the beginning of the tabbing order.

For more information on object reference variables, see Chapter 10 in the *Programmer's Guide*.

Example

NEW operator syntax:

```
LOCAL f
f=NEW EntryForm()
f.OPEN()
CLASS Entryform OF FORM
  this.view="Company.DBF"
  this.Fld1 = NEW Entryfield(this)
  this.Fld1.Top = 3
  this.Fld1.Left = 1
  this.Fld1.Width=20
  this.Fld1.Datalink = "Company->Company"
  this.Fld2 = NEW Entryfield(this)
  this.Fld2.Top = 5
  this.Fld2.Left = 1
  this.Fld2.Datalink = "Company->State_Prov"
  this.Fld2.Before = this.First
ENDCLASS
```

DEFINE object syntax:

```
DEFINE Entryfield Company OF THIS;
  Property Datalink "Company->Company",;
  WIDTH 20, Top 3, Left 1
DEFINE Entryfield State OF THIS;
  Property Datalink "Company->State_Prov",;
  Before this.First, Top 5, Left 1
```

See Also

ActiveControl, _curobj, BEFORE, NextObj, SetFocus()

FirstIndex

Returns the subscript character string for an element of an associated array.

Property of class

ASSOCARRAY

Description

Use FirstIndex when you want to step through the elements in an associated array object, starting from the first element in the array. Once you have issued FirstIndex and are positioned on the first element, use NextIndex() to step through the elements in order.

Note Elements in associated array objects are not necessarily stored in the order in which you add them to an array. This means you can't assume that the value returned by `FirstIndex` will be consistent, or that it will return the first item you added to the array. For more information, see `CLASS ASSOCARRAY`.

Example

The following example creates an associated array and displays its subscripts and contents.

```
aa = NEW ASSOCARRAY()
aa["San Francisco"] = "49ers"
aa["Los Angeles"] = "Rams"
x = aa.FirstIndex
DO WHILE .NOT. EMPTY(x)
    ? x, aa[x]           && display element subscript and contents
    x = aa.NextIndex(x)  && 'increments' index pointer
ENDDO
```

F

See Also

`IsIndex()`, `NextIndex()`

FocusBitmap

Specifies the graphic image to display in a pushbutton when the pushbutton has focus.

Property of class

`PUSHBUTTON`

Data type

Character

Default

The default for `FocusBitmap` is an empty string.

Description

Use `FocusBitmap` to indicate visually when a pushbutton is selected.

The `FocusBitmap` setting can take one of two forms:

- 1 `RESOURCE <resource id> <dll name>` specifies a bitmap resource and the DLL file that holds it. (A DLL file is a precompiled library of external routines and resources written in non-dBASE languages such as C and Pascal.) You can obtain the resource ID with a resource editor such as Resource Workshop.
- 2 `FILENAME <filename>` specifies a bitmap file (which usually has the file-name extension `.BMP`).

If you specify a character string with `Text` and an image with `FocusBitmap`, the image is displayed with the character string when the pushbutton has focus.

Follow

Note You can select a bitmap file with the Choose Bitmap dialog box. To access the Choose Bitmap dialog box, click the Tool button next to the FocusBitmap item in the Inspector.

Example

See DownBitmap for an example of FocusBitmap.

See Also

DEFINE, DisabledBitmap, DownBitmap, Enabled, UpBitmap

Follow

Determines if the display in a browse object follows a record to its new index order when a key field value is changed.

Property of class

BROWSE

Data type

Logical

Default

The default for Follow is true (.T.).

Description

Set Follow to true (.T.) to prevent the apparent disappearance of a record in a browse object when the records are in indexed order and the user changes a value in a key field.

When records are in indexed order, changing a key field value in a record changes the position of the record in the table. In a browse object, the changed record might disappear from the display view.

Example

NEW operator syntax:

```
this.Br1 = NEW BROWSE(this)
this.Br1.Alias="Contact"
this.Br1.Fields="CompCode,Contact"
this.Follow=.F.
this.Br1.Top=4
this.Br1.Left=3
this.Br1.Width=32
this.Br1.Height=12
```

DEFINE object syntax:

```
DEFINE BROWSE Br1 OF THIS;
PROPERTY;
Top 4, Left 3, Width 32, Height 12,;
```

```
Alias "Contact", Follow .F.;;
Fields "CompCode, Contact"
```

See Also

Modify

FontBold

Determines if an object displays characters in bold type.

F

Property of class

BROWSE, CHECKBOX, COMBOBOX, EDITOR, ENTRYFIELD, LISTBOX, PUSHBUTTON, RADIOBUTTON, RECTANGLE, SPINBOX, TABBOX, TEXT

Data type

Logical

Default

The default for FontBold is false (.F.) for all classes except EDITOR, ENTRYFIELD, PUSHBUTTON, RADIOBUTTON, SPINBOX, and TEXT.

Description

Use FontBold when you want to draw attention to text in an object prompt or text in an entry area.

For example, when you set FontBold property of a check box to true (.T.), the check box prompt is displayed in bold type. When you set the FontBold property of an entry field to true, the text in the entry field is displayed in bold type.

Example

See FontName for an example of FontBold.

See Also

FontItalic, FontName, FontSize, FontStrikeOut, FontUnderline

FontItalic

Determines if an object displays characters in italic type.

Property of class

BROWSE, CHECKBOX, COMBOBOX, EDITOR, ENTRYFIELD, LISTBOX, PUSHBUTTON, RADIOBUTTON, RECTANGLE, SPINBOX, TABBOX, TEXT

Data type

Logical

Default

The default for FontItalic is false (.F.).

Description

Use FontItalic when you want to give emphasis to text in an object prompt or text in an entry area.

For example, when you set the FontItalic property of a check box to true (.T.), the prompt is displayed in italic type. When you set the FontItalic property of an entry field to true, the text in the entry field is displayed in italic type.

Example

See FontName for an example of FontItalic.

See Also

FontBold, FontName, FontSize, FontStrikeOut, FontUnderline

FontName

Determines which font is applied to characters displayed in an object.

Property of class

BROWSE, CHECKBOX, COMBOBOX, EDITOR, ENTRYFIELD, LISTBOX, PUSHBUTTON, RADIOBUTTON, RECTANGLE, SPINBOX, TABBOX, TEXT

Data type

Character

Default

The default for FontName is MS Sans Serif.

Description

Use FontName to determine the visual appearance of characters in an object. For example, you can display characters in Arial format by inserting "Arial" into the Name property of a text object.

If the font you specify with FontName is not installed in your Windows environment, dBASE uses the default font, MS Sans Serif.

Note You can select a font with the Font dialog box, which displays all installed fonts. To access the Font dialog box, click on the Tool button next to the FontName item in the Inspector. You can also open the Font dialog box with the GETFONT() function.

Example**NEW operator syntax:**

```

Code = NEW ENTRYFIELD(this)
Code.Top = 5
Code.Left = 20
Code.Datalink = "Contact->CompCode"
Code.Height = 2
Code.FontBold = .T.
Code.FontSize = 12.00
Code.FontName = "Arial"
Code.FontItalic = .T

```

F**DEFINE object syntax:**

```

DEFINE ENTRYFIELD Code OF THIS;
Property;
Top 5, Left 20
Datalink "Contact->CompCode",;
Height 2,;
FontBold .T.,;
FontSize 12.00,;
FontName "Arial",;
FontItalic .T.

```

See Also

FontBold, FontItalic, FontSize, FontStrikeOut, FontUnderline, ScaleFontName

FontSize

Specifies the size of text characters in an object.

Property of class

BROWSE, CHECKBOX, COMBOBOX, EDITOR, ENTRYFIELD, LISTBOX, PUSHBUTTON, RADIOBUTTON, RECTANGLE, SPINBOX, TABBOX, TEXT

Data type

Numeric

Description

Use FontSize to adjust the size of characters displayed in an object. Character size is given in point size; the higher the point size, the larger the characters in the font.

Example**NEW operator syntax:**

```

this.Txt2 =NEW Text(this)
this.Txt2.Top = 5
this.Txt2.Left = 0

```

```
this.Txt2.Alignment = 4
this.Txt2.Text = "is here in '94"
this.Txt2.Height = 2
this.Txt2.Width = 50
this.Txt2.FontSize = 16
```

DEFINE object syntax:

```
DEFINE TEXT Txt2 OF THIS AT 5,4;
PROPERTY;
    Alignment 4;;
    Text "is here in '94",;
    Height 2;;
    Width 50;;
    FontSize 16
```

See Also

FontBold, FontItalic, FontName, FontStrikeOut, FontUnderline, ScaleFontSize

FontStrikeOut

Determines if an object displays characters in strikeout type.

Property of class

CHECKBOX, COMBOBOX, EDITOR, ENTRYFIELD, LISTBOX, PUSHBUTTON, RADIOBUTTON, RECTANGLE, SPINBOX, TABBOX, TEXT

Data type

Logical

Default

The default for FontStrikeOut is false (.F.).

Description

Use FontStrikeOut when you want to draw a line through text in an object.

For example, when you set the FontStrikeOut property of a check box to true (.T.), the check box prompt is displayed in strikeout type. When you set the FontStrikeOut property of an entry field to true, the text in the entry field is displayed in strikeout type.

Example

NEW operator syntax:

```
LOCAL f
f=NEW StrkOut()
f.OPEN()
CLASS StrkOut OF FORM
    this.ETF1 = NEW ENTRYFIELD(this)
    this.ETF1.Top = 2
```



```

this.ETF1.Left = 16
this.ETF1.Width=6
this.ETF1.Value = SPACE(6)
this.ETF1.FontStrikeOut=.T.
ENDCLASS

```

DEFINE object syntax:

```

* Form definition
DEFINE ENTRYFIELD ETF1 OF THIS;
  PROPERTY Top 2, Left 16, Width 6,;
  Value SPACE(6), FontStrikeOut .T.

```

F

See Also

FontBold, FontItalic, FontName, FontSize, FontUnderline

FontUnderline

Determines if an object displays characters in underlined type.

Property of class

CHECKBOX, COMBOBOX, EDITOR, ENTRYFIELD, LISTBOX, PUSHBUTTON, RADIOBUTTON, RECTANGLE, SPINBOX, TABBOX, TEXT

Data type

Logical

Default

The default for FontUnderline is false (.F.).

Description

Use FontUnderline when you want to underscore text in an object prompt or text in an entry area.

For example, when you set a the FontUnderline property of a check box to true (.T.), the check box prompt is underlined. When you set the FontUnderline property of an entry field to true, the text in the entry field is underlined.

Example

See FontName for an example of FontUnderline.

See Also

FontBold, FontItalic, FontName, FontSize, FontStrikeOut

Function

Formats text displayed in an object.

Property of class

ENTRYFIELD, SPINBOX, TEXT

Data type

Character

Default

The default for Function is an empty string.

Description

The argument you give to Function consists of one or more function symbols, which should not be separated by spaces, commas, or other characters.

dBASE recognizes the following function symbols:

(Encloses negative numbers in parentheses.
!	Converts letters to uppercase.
^	Displays numbers in exponential form.
\$	Inserts a dollar sign or the symbol defined with SET CURRENCY TO instead of leading spaces.
A	Restricts entry to alphabetic characters.
B	Left-aligns a numeric entry.
C	Displays CR (credit) after a positive number.
D	Displays and accepts entry of a date in the current SET DATE format.
E	Displays and accepts entry of a date in European (DD/MD/YY) format.
I	Centers the entry.
J	Right-aligns the entry.
L	Displays numbers with leading zeros.
M	Sequentially displays predefined options each time the user presses <i>SpaceBar</i> . Specify the options by listing them (separated with commas) after @M.
R	Inserts literal characters into the display without including them in the field.
S<n>	Limits the width of the display to <n> characters and horizontally scrolls the characters within it in <n> columns. <n> must be a positive integer.
T	Removes leading and trailing spaces from character values.
V<n>	Wraps a character string within a width specified by <n>. The width is measured in inches multiplied by 10 (for example, an <n> value of 15.5 is equivalent to 1.55 inches).
X	Displays DB (debit) after a negative number.
Z	Displays zeros as a blanks.

Example

See DataLink for an example of Function.

See Also

DEFINE, Picture

GetTextExtent()

Returns the length of a text string in units based on the font size of a parent form.

Property of class

TEXT

Description

Use GetTextExtent() to calculate a value for the Width property of a text object. For example, you need to adjust the Width setting of a text object each time a new character string is displayed, if the new string exceeds the current width of the text object.

GetTextExtent() compares the font of the text object with the font of the parent form and returns the width of the character string in *character units*. The width of a character unit is equal to the width of a column in the coordinate plane of the parent form. Placing the character unit width of the string in the Width property of the text object resizes the object to contain the string.

GetTextExtent() accepts the <text string> parameter, a character string.

Example

This example creates a form with two long entry fields. On leaving the second field, OnLostFocus property calls a procedure that initializes a string variable from the concatenation of both entry fields. The procedure then uses GetTextExtent() to return the length of the string (according to the current font) so that the width property of text object T1 can be changed to accommodate the concatenated string:

```

LOCAL f
f = NEW XFORM()
f.Open()

CLASS XFORM OF FORM
  this.HelpFile = ""
  this.Top =      2.76
  this.Height =   11.71
  this.Text = "Form"
  this.Left =     53.00
  this.Width =    41.80
  this.HelpId = ""

  DEFINE ENTRYFIELD FNAME OF THIS;
  PROPERTY;
    Top      3.00,,;
    Height   1.00,,;
    Value "          ",;
    Border .T.,;
    Left     6.00,,;
    Width    20.00

```

Group

```
DEFINE ENTRYFIELD LNAME OF THIS;
PROPERTY;
    OnLostFocus CLASS::XLENGTH;;
    Top          5.00;;
    Height       1.00;;
    Value "                ";;
    Border .T.;;
    Left         6.00;;
    Width        20.00

DEFINE TEXT T1 OF THIS;
PROPERTY;
    Top          7.00;;
    Height       1.00;;
    Border .F.;;
    Text "First and Last Name";
    ColorNormal "N/W";
    Left         6.00;;
    Width        25.00

PROCEDURE xLength
    ExtString =;
    TRIM(form.Fname.Value)+" "+form.Lname.Value
```

See Also

FontName, FontSize, LEN(), ScaleFontName, ScaleFontSize

Group

Starts an object group in the tabbing order of the parent form.

Property of class

CHECKBOX, PUSHBUTTON, RADIOBUTTON

Data type

Logical

Description

Use Group to determine if an object is part of a group within which the user can move focus with the arrow keys.

When you use Group to group two or more radio buttons together, the user can select only one button from the group at a time. This is useful when you want the user to select one out of several values.

To create a group, set the Group property of the first object in the group to true. For all other objects that belong to the group, set Group to false (.F.). The next object with a Group setting of true begins another group.

Example

NEW operator syntax:

```

Answer=""
Yes = New RADIOBUTTON(this)
Yes.Text = "Yes"
Yes.DataLink = "Answer"
Yes.Group = .F.
No = New RADIOBUTTON(this)
No.Top = 5
No.Text = "No"
No.DataLink = "Answer"

```

DEFINE object syntax:

```

Answer=""
DEFINE RADIOBUTTON Yes OF THIS;
  PROPERTY;
    Text "Yes",;
    DataLink "Answer",;
    Group .F.
DEFINE RADIOBUTTON No OF THIS;
  PROPERTY;
    Top 5,,;
    Text "No",;
    DataLink "Answer"

```

G

See Also`_curobj`, `ID`, `TabStop`

Grow()

Adds elements to an array object.

Property of class

ARRAY

Description

Use the `Grow()` method to add an element, row, or column to an array object. All new elements have the value logical `.F.` The `Grow()` method is similar to the `AGROW()` function.

`Grow()` accepts one required `<expN>` parameter, which can be 1 or 2. If you attempt to assign a value other than 1 or 2, dBASE displays the error message "Invalid function argument".

- If you set `<expN>` to 1, `Grow()` adds a single element to a one-dimensional array object or a row to a two-dimensional array object.

- If you set `<expN>` to 2, `Grow()` adds a column to the array object. If the array object is one-dimensional, `Grow()` makes it two-dimensional; it moves existing elements to the first column and fills the second column with .F. values.

If the array object is two-dimensional, `Grow()` adds a column at the end of the array and fills the new elements with .F. values.

See `Insert()` for another way to add elements to an array object. To add multiple rows or columns to an array object, use `Resize()`.

Example

```
USE Customer.DBF
* Initialize an array object
ObjArr=NEW ARRAY(RECCOUNT())
* Fill 1-dimensional array with values
* from Name field of Customer.DBF
GO TOP
FOR i=1 TO RECCOUNT()
    ObjArr[i]=Customer->Name
    SKIP
NEXT i
* Use GROW() to add a second column to the
* existing array and enter Ytd_Sales data
* in new second column.
ObjArr.GROW(2)
GO TOP
FOR i=1 TO RECCOUNT()
    ObjArr[i,2]=Customer->Ytd_Sales
    SKIP
NEXT i
* Display Contents of 2-dimensional array
FOR i=1 TO RECCOUNT()
    ? ObjArr[i,1], ObjArr[i,2]
NEXT i
```

See Also

`AGROW()`, `Insert()`, `Resize()`

Header3D

Specifies whether the top and left portions of a browse object appear raised (three-dimensional).

Property of class

BROWSE

Data type

Logical

Default

The default for Header3D is true (.T.).

Description

Header3D affects the appearance of the top and left sides of a browse window when ShowHeading and/or ShowRecNo (respectively) are true. If Header3D is true, the Heading and Record number appear three-dimensional, making it easier to see that they don't represent values contained in the table. If ShowHeading and ShowRecNo are false, Header3D has no effect.

Example

The following example shows a form with browse windows having three different display formats.

```
f=NEW FORM()
f.View = "CLIENTS.DBF"
f.Width=80
b = NEW BROWSE(f)
  b.Alias = "clients"
  b.ColorNormal = "WindowText/Window"
  b.colorHighLight = "WindowText/Window"
  b.Header3D=.T.          && 3D top and left

b3d = NEW BROWSE(f)
  b3d.Alias = "clients"
  b3d.Left = b.left+20
  b3d.ColorNormal = "WindowText/Window"
  b3d.colorHighLight = "WindowText/Window"
  b3d.Header3D=.F.        && Normal top and left

b3da = NEW BROWSE(f)
  b3da.Alias = "clients"
  b3da.Left = b3d.left+20
  b3da.ColorNormal = "WindowText/Window"
  b3da.colorHighLight = "WindowText/Window"
  b3da.Header3D=.T.        && Ignored because
  b3da.ShowRecNo=.F.        && ShowRecNo and
  b3da.ShowHeading=.F.      && ShowHeading are false
f.open()
```

H

See Also

ShowHeading, ShowRecNo

Height

Determines the height of an object.

Property of class

BROWSE, CHECKBOX, COMBOBOX, EDITOR, ENTRYFIELD, FORM, IMAGE, LISTBOX, OLE, PUSHBUTTON, RADIOBUTTON, RECTANGLE, SCROLLBAR, SPINBOX, TABBOX, TEXT

Data type

Numeric

Description

Use the Height property in combination with the Width property to adjust the size of an object. The setting you give to Height determines the distance from the base of an object to its top.

Each unit of the value you assign to Height is the average height of characters in the active font of the parent form. For example, if you set the Height property of an editor object to 17.5, the editor object is 17.5 characters high.

Since Height accepts integer or non-integer values, you can adjust the height of an object to a high degree of precision.

Example

NEW operator syntax:

```
Code = NEW ENTRYFIELD(this)
Code.Top = 5
Code.Left = 1
Code.Datalink = "Contact->CompCode"
Code.Height = 2
Code.Left = 20
* Because the default height for an
* entryfield is 1.17, setting height to
* 2 in this example merely offers additional
* space around the field value.
```

DEFINE object syntax:

```
DEFINE ENTRYFIELD Code OF THIS AT 5,1;
Property;
Datalink "CompCode",;
Height 2, Left 20
```

See Also

Left, Top, Width

HelpFile

Identifies a Windows Help file (.HLP) that contains context-sensitive Help topics.

Property of class

BROWSE, CHECKBOX, COMBOBOX, EDITOR, ENTRYFIELD, FORM, LISTBOX, MENU, PUSHBUTTON, RADIOBUTTON, SCROLLBAR, SPINBOX, TABBOX

Data type

Character

Default

The default for HelpFile is an empty string.

Description

Use HelpFile in combination with HelpID to connect an object to a topic in a Windows Help file. For example, HelpID might identify a Help topic that gives instructions on using a combo box.

Enter the name of the Help file into the HelpFile property, and set the HelpID property to the Help keyword of the topic. When the user gives focus to the object and presses *F1*, Windows Help opens the Help file and displays the Help topic.

For information on creating Help files and Help topics, see your Windows documentation.

Note Any subroutine you specify with the OnHelp property overrides the values you assign to HelpFile and HelpID.

Example

NEW operator syntax:

```
Company = NEW ENTRYFIELD(this)
Company.Top = 3
Company.Left = 1
Company.Datalink = "Company"
Company.HelpFile = "DataEntr"
Company.HelpId = 100
```

DEFINE object syntax:

```
DEFINE ENTRYFIELD Company OF THIS AT 3,1;
Property;
Datalink "Company",;
HelpFile "DataEntr",;
HelpId 100
```

See Also

HELP, HelpID, OnHelp, SET HELP



HelpID

Specifies the Help keyword of a Windows Help topic.

Property of class

BROWSE, CHECKBOX, COMBOBOX, EDITOR, ENTRYFIELD, FORM, LISTBOX, MENU, PUSHBUTTON, RADIOBUTTON, SCROLLBAR, SPINBOX, TABBOX

Data type

Character

Default

The default for HelpID is an empty string.

Description

Use HelpID in combination with HelpFile to connect an object with a topic in a Windows Help file (.HLP). For example, HelpFile might contain a Help topic that gives instructions on using the objects in a form.

Set the HelpID property to the Help keyword of the topic and enter the name of the Help file into the HelpFile property. When the user gives focus to the object and presses *F1*, the Help system opens the Help file and displays the Help topic.

Note Any subroutine you specify with the OnHelp property overrides the values you assign to HelpID and HelpFile.

Example

See HelpFile for an example of HelpId.

See Also

HELP, HelpFile, OnHelp, SET HELP

hWnd

Specifies an object handle.

Property of class

BROWSE, CHECKBOX, COMBOBOX, EDITOR, ENTRYFIELD, FORM, IMAGE, LISTBOX, OLE, PUSHBUTTON, RADIOBUTTON, RECTANGLE, SCROLLBAR, SPINBOX, TABBOX, TEXT

Data type

Numeric

Default

There is no default for hWnd; Windows generates the value automatically.

Description

An object handle is a unique numeric value that Windows generates automatically for each object you create. External functions written in other languages such as C, Pascal, or ASM use this handle to identify the object. For example, you can pass the object handle of a form as a parameter to an external function, perhaps allowing the function to open or close the form.

Such external functions are held in external C libraries such as Windows API or a customized DLL file you create yourself. For more information on external functions and on DLL files, see EXTERN, LOAD DLL, and Chapter 25 in the *Programmer's Guide*.

The hWnd property is read-only.

Example

```
DEFINE FORM Entry FROM 2,2 TO 30,70
DEFINE ENTRYFIELD Company OF THIS;
PROPERTY;
    Datalink "Company"
    FormHandle=Entry.hwnd      && e.g. 14260
    CompanyHandle=Entry.Company.hwnd && e.g. 14348
```

See Also

EXTERN, LOAD DLL

Icon

Specifies an icon format file (.ICO) or resource that displays when a form is minimized.

Property of class

FORM

Data type

Character

Default

The default for Icon is an empty string.

Description

Use Icon to specify an image to be used when a form is minimized. You can specify a resource (generally from a .DLL or .VBX file) or a file name.

Example

In the following example, the form is minimized when it opens, and an icon from a specified DLL file is displayed:

```
f = NEW Form()
f.Icon = "RESOURCE #20 C:\MYAPP\MYAPP.DLL"
f.WindowState=1
```

See Also

Minimize, WindowState

ID

Identifies an object with a numeric value.

Property of class

BROWSE, CHECKBOX, COMBOBOX, EDITOR, ENTRYFIELD, IMAGE, LISTBOX, MENU, OLE, POPUP, PUSHBUTTON, RADIOBUTTON, SCROLLBAR, SPINBOX, TABBOX, TEXT

Data type

Numeric

Default

The default for ID is -1.

Description

Use ID to give a unique identifier to an object.

For example, when the user chooses a pushbutton, the form is submitted, executing the subroutine or codeblock you assigned with the OnSelection property or the ON SELECTION FORM command. The value in ID is passed to the subroutine or codeblock, identifying the pushbutton that submitted the form.

If you wish, you can use the ID value to make branching decisions based on which object executed the OnSelection subroutine. To be sure that ID correctly identifies the object it represents, give it a unique value for each object.

The ID values for menu objects are generated by dBASE automatically, and are read-only.

Example

NEW operator syntax:

```
Date = NEW ENTRYFIELD(this)
Date.Datalink = "Clients->Baldate"
Date.Id = 10
```

DEFINE object syntax:

```
DEFINE ENTRYFIELD Date OF THIS;
  PROPERTY Datalink "Clients->Baldate",;
  ID 10
```

See Also

OnSelection, ON SELECTION FORM, PARAMETERS

Initiate()

Starts a conversation with an external application.

Property of class

DDELINK

Description

Use Initiate() to open a channel of communication (known as a *DDE link*) between dBASE and an external Windows application.

Use a DDE link to exchange data and instructions with another application. For example, a data-exchange program might use Initiate() to establish a link with Quattro Pro for Windows and open one of its spreadsheet files, then copy data from the spreadsheet to a dBASE table.

If the external application is not already running when you execute Initiate(), dBASE tries to start the application before establishing the DDE link. If the attempt is unsuccessful, Initiate() returns a value of false (.F.).

Initiate() requires two parameters:

- <server> is the main executable file of the server application.
- <topic> is the name of the data file to open in the server session.

To terminate the DDE link, use the Terminate() method.

Example

```
PUBLIC LinkObj
LinkObj = NEW DDELINK()
LinkObj.Initiate("QPW", "Demo.WB1");
    && Attempt to establish a link with a;
    QPW server
```

See Also

Advise(), Execute(), OnNewValue, Peek(), Poke(), Server, Terminate(), TimeOut, Topic, Unadvise()

Insert()

Inserts a .F. value into an element in a one-dimensional array object, or inserts .F. values into a row or column of elements in a two-dimensional array object.

Property of class

ARRAY

Description

Use Insert() to insert .F. values into selected elements in an array object without changing the size of the array object. Insert() does the following:

- Inserts an element in a one-dimensional array, or inserts a row or column in a two-dimensional array
- Moves all remaining elements toward the end of the array (down if a row is inserted, to the right if an element or column is inserted)
- Inserts .F. values in the newly created element or elements

Insert() accepts two parameters:

- *<position expN>* specifies the number of the element at which to insert the new element when the array object is one-dimensional. Specifies a row or column when the array object is two-dimensional.
- 2 determines if Insert() inserts a row or a column in a two-dimensional array object. If you specify 2, it inserts a row; otherwise, it inserts a column. Insert() ignores this parameter if the array object is one-dimensional.

For information about inserting elements by moving remaining elements toward the *beginning* of the array, see Delete(). For information about replacing elements without moving remaining elements, see Fill(). To change a one-dimensional array to two-dimensional, use Grow() or Resize().

Example

```
USE Animals.DBF
* Initialize an array object
ObjArr=NEW ARRAY(RECCOUNT(),3)
* Fill array with values from Animals.DBF
COPY TO ARRAY ObjArr FIELDS Name, Area, Weight
* Use Insert() to replace weight values in
* column 3 with .F..
ObjArr.Insert(3,2)
* Display results to Command window results pane
FOR i=1 TO RECCOUNT()
? ObjArr[i,1],ObjArr[i,2],ObjArr[i,3]
NEXT i
```

See Also

Add(), Delete(), Resize()

IsIndex()

Returns a logical value that indicates if the specified character expression is a subscript of an element in an associated array.

Property of class

ASSOCARRAY

Description

Use IsIndex(<expC>) to determine if an associated array contains an element with a subscript of <expC>. This might be useful if you want to add an item to the array only if it is not already there, or if you want to remove an element that has been added to the array.

Example

The following example creates an associated array based on an existing table. It then uses IsIndex() to see if a specific element is in the array. If it isn't, it adds it to the array, and to the table that was used to populate the array.

```
aa = NEW ASSOCARRAY()
USE customer
SCAN
    cName = TRIM(name)
    aa[cName] = city                && Create element for each record
ENDSCAN
NewName = "Just Testing"
NewCity = "Anywhere"
IF aa.IsIndex(NewName) = .F. && No matching element in array
    APPEND BLANK
    REPL name WITH NewName, ;
        city WITH NewCity          && Add record to table
    aa[NewName] = NewCity          && Add element to array
ENDIF
```

See Also

FirstIndex, NextIndex(), RemoveKey()

IsRecordChanged()

Returns a logical value that indicates whether the current record in the append buffer, created with BeginAppend(), has been modified.

Property of class

Form

Data type

Logical

Description

Use `IsRecordChanged()` to determine whether a user has modified a new record created with `BeginAppend()`. For example, if a user tries to close the new record without saving it, check for `IsRecordChanged()`. If it's true, you can prompt the user to confirm they want to abandon their changes.

Example

See `BeginAppend()` for an example.

See Also

`AbandonRecord()`, `BeginAppend()`, `SaveRecord()`

Key

Executes a subroutine when the user enters a keystroke in an entry field.

Property of class

ENTRYFIELD

Data type

Function pointer or codeblock

Description

Use `Key` to evaluate and modify each character that the user enters in an entry field or browse object. For example, an entry field that accepts passwords might convert each character the user inputs to an asterisk.

Like other event properties, the `Key` property accepts

- Functions
- Procedures
- Codeblocks

For information on these subroutines, see Chapter 4 in the *Programmer's Guide*. For information on writing event-handling routines, see Chapter 14 in the *Programmer's Guide*.

`Key` sends two parameters to its subroutine:

- `<nChar>` is the ASCII decimal number of the new character entered by the user.
- `<nPosition>` is the position of the new character in the string.

For example, when the third character entered in an entry field is "a", the `<nChar>` parameter is 97 and the `<nPosition>` parameter is 3.

The subroutine you create for `Key` must return a numeric value or a logical value. A numeric value is interpreted as the ASCII decimal number of a character, which automatically replaces the character input by the user. A logical value is interpreted as a decision to accept or reject the character input by the user.

Note You can write a subroutine with the Procedure Editor, a window in which you enter dBASE program code. To access the Procedure Editor, click on the Tool button next to the Key item in the Inspector.

Example

NEW operator syntax:

```
Company = NEW ENTRYFIELD(this)
Company.Datalink = "Company->Company"
Company.Key = AnyKey
* A Function Anykey will be called
FUNCTION ANYKEY(nChar, nPosition)
* process the keystroke
RETURN nChar + 1
```

DEFINE object syntax:

```
DEFINE ENTRYFIELD Company OF THIS;
PROPERTY;
Datalink "Company->Company", ;
Key AnyKey
```

See Also

OnChange, OnGotFocus

K

Keyboard()

Passes a character string to an edit control, simulating typed user input.

Property of class

BROWSE, COMBOBOX, EDITOR, ENTRYFIELD, SPINBOX

Description

Use Keyboard() when you want to pass text to an entry control as if the user had typed it in manually. For example, if you are writing a tutorial, you can ask the user for information, then use Keyboard() to demonstrate where the user would type the information in an editor object on a form.

Example

The following example illustrates using Keyboard() to simulate the typing of text in an entry field.

```
f = new KBDFORM()
f.Open()
CLASS KBDFORM OF FORM
this.Text = "Form"
this.Top = 0
this.PageNo = 0
this.Width = 80
```

```

this.ColorNormal = "BtnText/BtnFace"
this.OnOpen = CLASS::FORM_ONOPEN
DEFINE ENTRYFIELD EF1 OF THIS;
    PROPERTY;
        Border .T.,;
        ColorHighLight "WINDOWTEXT/WINDOW",;
        Top 6,,;
        PageNo 1,,;
        Width 30,,;
        ColorNormal "WINDOWTEXT/WINDOW",;
        Height 1,,;
        Left 12
    Procedure Form_OnOpen
    this.ef1.setfocus()
    this.ef1.keyboard("Type your name here")

ENDCLASS

```

See Also

Paste()

Left

Specifies the position of the left border of an object relative to its parent form, or the position of a form relative to the desktop.

Property of class

BROWSE, CHECKBOX, COMBOBOX, EDITOR, ENTRYFIELD, FORM, IMAGE, LINE, LISTBOX, OLE, PUSHBUTTON, RADIOBUTTON, RECTANGLE, SCROLLBAR, SPINBOX, TABBOX

Data type

Numeric

Description

Use the Left property in combination with the Top property to position an object in a form or position a form in the desktop. (The Top property specifies the position of the top border.) Left and Top accept integer or non-integer values.

If you use the FROM or AT clauses of the DEFINE command *and* specify a value for the Left property, dBASE positions the object with the Left property.

Each unit of the value you assign to Left is the average width of characters in the active font of the parent form. For example, if you set the Left property of a line to 20, the left end of the line is positioned 20 characters from the left edge of the form.

Example

See Height for an example of Left.

See Also

CLASS FORM, Bottom, Height, Right, ScaleFontName, ScaleFontSize, Top, Width

LineNo

Sets the current line in an editor object.

Property of class

EDITOR

Data type

Numeric

Default

The default for LineNo is 1.

Description

Use LineNo to move the cursor to a specified line in an editor object.

When you move the cursor to a line, the cursor maintains its original column position if the new line is long enough. If the line is not long enough, the cursor is placed at the end of the line.

L

Example

```
LOCAL f
f=NEW AdjustEdit()
f.OPEN()
CLASS AdjustEdit OF EDITOR
    this.LineNo = 15 && Move cursor to 15th line.
ENDCLASS
```

See Also

Wrap

LinkFileName

Identifies which OLE document file (if any) is linked with the current OLE field when that field is displayed in an OLE viewer.

Property of class

OLE

Data type

Character

Default

The default for LinkFileName is an empty string.

Description

Use LinkFileName to identify which OLE document file is linked with the current OLE field when that field is displayed in an OLE viewer.

An OLE viewer displays an OLE document (sometimes called an OLE object). An OLE document can be a graphic image, a document created by a word processor, or any other data object created by an external application. This external application is known as the *OLE server*. For example, a bitmap file (.BMP) created in Paintbrush can be an OLE document, and Paintbrush can be an OLE server, if you link the bitmap file to an OLE field with the Cut and Paste Link commands of the Edit menu. LinkFileName contains the name of this file.

LinkFileName is a read-only property.

Example

The following program creates a form that displays an OLE object and a pushbutton. The OnClick subroutine of the pushbutton uses LinkFileName to display the name of the document file in a message box.

```
LOCAL f
f = NEW XFORM()
f.Open()

CLASS XFORM OF FORM
  this.HelpFile = ""
  this.Top =      0.47
  this.Height =   26.00
  this.Text = "Form"
  this.View = "PICTURES.QBE"
  this.MousePointer = 1
  this.Left =     23.00
  this.Width =    101.20
  this.HelpId = ""

  DEFINE OLE OLE1 OF THIS;
  PROPERTY;
  DataLink "PICTURES->BITMAPOLE",;
  Top      1.00,;
  Height   21.00,;
  Border .T.,;
  Left     1.00,;
  Width    99.00

  DEFINE PUSHBUTTON BUTTON1 OF THIS;
  PROPERTY;
  Top      23.00,;
  Height   2.00,;
  OnClick CLASS::BUTTON1_ONCLICK,;
  Text "Push me to get name of bitmap file.",;
  ColorNormal "N/W",;
  Default .T.,;
```

```

        Left      36.00,;
        Width     31.00

    Procedure BUTTON1_OnClick
        MSGBOX(form.OLE1.LinkFileName, "Here's the file...")
    ENDCLASS

```

See Also

OleType, ServerName

Maximize

Determines if a form can be maximized.

Property of class

FORM

Data type

Logical

Default

The default for Maximize is true (.T.).

Description

Like other windows, a form normally has a Minimize button and a Maximize button at the upper right corner. Choosing Minimize reduces the form to an icon, and choosing Maximize expands the form to fill the work area of the desktop.

Setting the Maximize property to false (.F.) removes the Maximize button, preventing the user from maximizing the form. This is useful when you want portions of the work area to be visible at all times, as when other forms are open or when the user needs access to the Command window.

Note When you set the MDI property of a form to true, the Maximize setting is ignored and the user can always maximize the form.

Example

NEW operator syntax:

```

f=NEW EntryForm()
f.OPEN()
CLASS EntryForm OF FORM
    this.MDI = .F.
    this.Maximize = .F.
    this.Minimize = .F.
ENDCLASS

```

M

DEFINE object syntax:

```
DEFINE FORM Entry FROM 1,1 TO 30,35;  
  Property;  
    MDI = .F.,;  
    Maximize .F.,;  
    Minimize .F.
```

See Also

MDI, Moveable, Minimize, Sizeable

MaxLength

Specifies the scrolling width of an entry field.

Property of class

ENTRYFIELD

Data type

Numeric

Description

Use MaxLength to set a limit on how many columns the user can scroll text to the left in an entry field.

For example, the user can input data into an entry field even if it isn't linked to a field. It might be necessary to limit the number of characters the user can input so the contents of the entry field can be placed in a character field later. Setting the MaxLength property to a number at or below the length of the field ensures that the user can't enter too long of a string.

The columns are as wide as the average width of characters of the active font.

Example

NEW operator syntax:

```
Company = NEW ENTRYFIELD(this)  
Company.Top = 3  
Company.Left = 1  
Company.Datalink = "Company"  
Company.MaxLength = 10
```

DEFINE object syntax:

```
DEFINE ENTRYFIELD Company OF THIS;  
  AT 3,1;  
  Property;  
    Datalink "Company",;  
    MaxLength 10
```

See Also

Function, Height, Picture, ScaleFontName, ScaleFontSize, Width

MDI

Determines if a form conforms to the Windows Multiple Document Interface (MDI) standard.

Property of class

FORM

Data type

Logical

Default

The default for MDI is true (.T.).

Description

MDI (Multiple-Documents Interface) is a Windows feature that lets you open multiple *document windows* within the application window. An application such as dBASE uses MDI to manage multiple documents or multiple views of the same document within the main application window. These views or documents appear in separate document windows. If you want your form to appear as a document window, set MDI to true.

The following lists some of the characteristics of document windows:

- Like application windows, they are moveable and sizeable
- They are listed on the Windows menu even when they are active.
- They have a Control-menu box, Maximize and Minimize buttons, and a title bar that contains the window name.
- When they are active, their menus replace the menus in the main menu bar. (When the MDI property is false (.F.), the menus are displayed in the menu bar of the form instead.)

When you set the MDI property to true:

- The form is a child of the application window.
- The Minimize setting is ignored, so the user can always minimize the form.
- The Maximize setting is ignored, so the user can always maximize the form.
- The Sizeable setting is ignored, so the user can always resize the form.
- The Moveable setting is ignored, so the user can always move the form.
- The SysMenu setting is ignored, so the Control menu (also known as the *system menu*) is always accessible from the *Control Menu box*, a button at the upper left corner of a form.

M

- The shortcut keystroke that closes the form is *Ctrl+F4* instead of *Alt+F4*; that is, *Alt+F4* closes the application window, while *Ctrl+F4* closes the form.

An MDI form is modeless by definition, since focus can be removed from it. Consequently, you can't open an MDI form with the `READMODAL()` function or the `ReadModal()` method.

For more information on MDI, consult your Windows documentation.

Example

NEW operator syntax:

```
LOCAL f
f = NEW EntryForm()
f.MDI = .F.
f.Open()
```

DEFINE object syntax:

```
DEFINE FORM EntryForm ;
    Property MDI .F.
OPEN FORM EntryForm
```

See Also

CLASS FORM, Maximize, Menu, Minimize, Moveable, Sizeable, SysMenu

MenuFile

Assigns a predefined menu system to a form.

Property of class

FORM

Data type

Character

Default

The default for MenuFile is an empty string.

Description

Use MenuFile to specify a menu definition file (.MNU) for a form.

A menu definition file is a text file that contains dBASE code for generating menus. dBASE displays the menus in the form or the Application window (depending on the MDI setting) when you open the form.

Note You can create a menu definition file with the Menu Designer, which you access by clicking the Tool button next to the MenuFile property in the Inspector.

Example**NEW operator syntax:**

```

LOCAL f
f=NEW Equipmnt()
CLASS Equipmnt OF FORM
    this.MDI = .F.
    this.ColorNormal = "W/B"
    this.Text = "Flight Equipment Management"
    this.Width = 78.00
    this.Height = 40.00
    this.View = "Equipmnt.QBE"
    this.MenuFile = "Equipmnt.MNU"
* See Equipmnt.WFM and Equipmnt.MNU on the
* DBASEWIN\SAMPLES directory.

```

DEFINE object syntax:

```

DEFINE FORM Equipmnt;
    PROPERTY;
    MDI .F.,;
    ColorNormal "W/B",;
    Width 78, Height 40,;
    View "Equipmnt.QBE",;
    MenuFile "Equipmnt.MNU"

```

See Also

CLASS MENU

M

Minimize

Determines if a form can be minimized.

Property of class

FORM

Data type

Logical

Default

The default for Minimize is true (.T.).

Description

Like other windows, a form normally has a Minimize button and a Maximize button at the upper right corner. Choosing Minimize reduces the form to an icon, and choosing Maximize expands the form to fill the work area of the desktop.

Setting the Minimize property to false (.F.) removes the Minimize button, preventing the user from minimizing the form. This is useful when you want the form to be visible any

time it is open, as when it contains objects or information that must be dealt with immediately.

Note When you set the MDI property of a form to true, the Minimize setting is ignored and the user can always minimize the form.

Example

See Maximize for an example of Minimize.

See Also

MDI, Moveable, Maximize, Sizeable

Mode

Specifies the format of a browse object.

Property of class

BROWSE

Data type

Numeric

Default

The default for Mode is 0 (Browse).

Description

Use Mode to determine how data is displayed in a browse object.

You can assign Mode one of three values:

Mode value	Display format	Shows
0 (Browse)	Row-and-column display	Multiple records
1 (Form Edit)	Form layout	Single record
2 (Columnar Edit)	Single-column layout	Single record

If the Toggle property is true (.T.), the user can switch between all of these formats by pressing F2.

Example

NEW operator syntax:

```
CompanyBrowse = NEW BROWSE(this)
CompanyBrowse.Top = 3
CompanyBrowse.Left = 1
CompanyBrowse.Height = 10
CompanyBrowse.Width = 59
CompanyBrowse.Alias = "Company"
```

```
CompanyBrowse.Mode = 2
CompanyBrowse.Fields="Company,Ytd_Sales;
:H='Sales,Ytd'"
```

DEFINE object syntax:

```
DEFINE BROWSE CompanyBrowse OF THIS;
FROM 3,1 TO 13,60;
PROPERTY;
Alias "Company", Mode 2;;
Fields "Company,Ytd_Sales:H='Sales, Ytd'"
```

See Also

BROWSE, CLASS BROWSE, EDIT

Modify

Determines if the user can alter records in a browse or editor object.

Property of class

BROWSE, EDITOR

Data type

Logical

Default

The default for Modify is true (.T.).

M

Description

Set Modify to false (.F.) when you want to prevent users from changing records in a browse object. For example, a card catalog program might manage records on books in a library, and each record might hold a book description in a memo field. The program could let users view the descriptions in a browse object, but prevent users from altering them.

Example

NEW operator syntax:

```
this.Br1=NEW BROWSE(this)
this.Br1.Alias="Contact"
this.Br1.Modify=.F.
this.Br1.Top=4
this.Br1.Left=3
this.Br1.Width=32
this.Br1.Height=12
```

DEFINE object syntax:

```
DEFINE BROWSE Br1 OF THIS;  
PROPERTY;  
  Alias "Contact", Modify .F.,;  
  Top 4, Left 3, Width 32, Height 12
```

See Also

Append, Delete

MousePointer

Changes the appearance of the mouse pointer.

Property of class

BROWSE, CHECKBOX, COMBOBOX, EDITOR, ENTRYFIELD, FORM, IMAGE, LISTBOX, OLE, PUSHBUTTON, RADIOBUTTON, RECTANGLE, SCROLLBAR, SPINBOX, TEXT

Data type

Numeric












Default

The default for MousePointer is 0 (Default).

Description

Use MousePointer to provide a visual cue when the user moves the mouse pointer over an object. For example, one pointer style might mean an object is disabled, while another pointer style might mean the object is ready for input.

You can specify the following settings for MousePointer:

0 (Default)	N/A	6 (Size NESW)	
1 (Arrow)		7 (Size S)	
2 (Cross)		8 (Size NWSE)	
3 (I-Beam)		9 (Size E)	
4 (Icon)		10 (UpArrow)	
5 (Size)		11 (Wait)	

Example

NEW operator syntax:

```

DEFINE BROWSE Br1 OF THIS
  this.MousePointer=1
  this.Br1.Top=4
  this.Br1.Left=3
  this.Br1.Width=32
  this.Br1.Height=12

```

DEFINE object syntax:

```

DEFINE BROWSE Br1 OF THIS FROM 4,3 TO 16,35;
  PROPERTY MousePointer 1

```

See Also

OnMouseMove

Move()

Repositions and resizes an object in its parent form.

Property of class

BROWSE, CHECKBOX, COMBOBOX, EDITOR, ENTRYFIELD, FORM, IMAGE, LISTBOX, RADIOBUTTON, PUSHBUTTON, RECTANGLE, SCROLLBAR, SPINBOX, TABBOX, TEXT

Description

Use the Move() method to change an objects position and dimensions.

Move() uses four parameters:

- *<left column expN>* sets the position of the left border, affecting the value in the Left property.
- *<top row expN>* sets the position of the top border, affecting the value in the Top property.
- *<width expN>* sets the width, affecting the value in the Width property.
- *<height expN>* sets the height, affecting the value in the Height property.

The units of distance you specify for Move() are based on the average width and height of characters in the active font.

Example

NEW operator syntax:

```

this.Lbl1 = NEW LISTBOX(THIS)
  this.Lbl1.DataSource="FIELD Animals->Name"
  this.Lbl1.Top=4
  this.Lbl1.Left=6
  this.Lbl1.Width=20
  this.Lbl1.Height=12
  this.Lbl1.OnRightMouseDown {;Form.Move(6,10,15,5)}

```

M

DEFINE object syntax:

```

DEFINE LISTBOX LB1 OF THIS;
  PROPERTY;
    DataSource "FIELD Animals->Name",;
    Top 4,;
    Left 6,;
    Width 20,;
    Height 12,;
    OnRightMouseDown {;Form.Move(6,10,15,5)}

```

See Also

Height, Left, ScaleFontName, ScaleFontSize, Top, Width

Moveable

Determines if a form can be moved with the mouse.

Property of class

FORM

Data type

Logical

Default

The default for Moveable is true (.T.).

Description

When you set the Moveable property of a form to true, the form has a title bar at the top. The user can place the mouse over the title bar, hold the left mouse button down, and drag the form to another location. A form can be moved only if it has a title bar.

When you set the MDI property of a form to true, the Moveable setting is ignored and the user can always move the form.

Example

NEW operator syntax:

```

LOCAL f
f=NEW EntryForm()
f.OPEN()
CLASS EntryForm OF FORM
  this.Moveable = .F.
ENDCLASS

```

DEFINE object syntax:

```
DEFINE FORM EntryForm;
    Property Moveable .F.
OPEN FORM EntryForm
```

See Also

Left, Sizeable, Top

Multiple

Determines if more than one item in a list box can be selected.

Property of class

LISTBOX

Data type

Logical

Default

The default for Multiple is false (.F.).

Description

When you set the Multiple property to true (.T.), the user can choose any number of the list box items (or none at all), and the list box is said to be *multiple-choice*. If you set Multiple to false, the user can choose only one prompt from the list box.

Multiple-choice list boxes are useful for allowing the user to make several choices at once. For example, a program written for a Human Resources department might allow the user to select several job applicants for interviewing. This program could let the user choose one or more applicants from a multiple-choice list box. (Each chosen prompt is tagged with a checkmark.)

Note To evaluate which prompts were chosen, use LISTSELECTED() or Selected().

Example

NEW operator syntax:

```
LB1 = NEW LISTBOX(this)
LB1.DataSource = "FIELD COMPANY->COMPANY"
LB1.Multiple = .T.
LB1.OnRightMouseDown = Checked
* Multiple companies can be chosen
```

DEFINE object syntax:

```
DEFINE LISTBOX LB1 OF THIS;
    PROPERTY;
        DataSource "FIELD COMPANY->COMPANY",;
        Multiple .T., OnRightMouseDown Checked
```

M

```

PROCEDURE Checked
FOR i=1 TO Form.LB1.Count()
    ? Form.LB1.Selected(i)
NEXT i
RETURN

```

See Also

Count(), LISTSELECTED(), Selected()

Name

A read-only property that specifies the name of an object.

Property of class

BROWSE, CHECKBOX, COMBOBOX, EDITOR, ENTRYFIELD, IMAGE, LINE, LISTBOX, MENU, OLE, PUSHBUTTON, RADIOBUTTON, SCROLLBAR, SPINBOX, TABBOX, TEXT

Data type

Character

Default

The default for Name is the name dBASE assigns to the object when you create it. For example, dBASE automatically assigns "BUTTON1" to the Name property of the first pushbutton you create with the Form Designer.

Description

Use the Name property to name an object.

When you create an object with the DEFINE command, you specify a name for the object with the *<object name>* parameter. For more information, see DEFINE.

When you create an object with the NEW operator, you can specify a name for the object with the second (optional) parameter. For example, the following commands create a form and an image object to display in it:

```

MyForm = NEW FORM()
MyImage = NEW IMAGE(MyForm, "OurImage")

```

The Name property of the new image object contains "OurImage".

Example

NEW operator syntax:

```

XXX = NEW LISTBOX(this)
XXX.DataSource = "FIELD COMPANY->Zip_P_Code"
XXX.Name = "ZipList"
* ZipList overrides XXX as the name of the object

```

DEFINE object syntax:


```

DEFINE LISTBOX XXX OF THIS;
PROPERTY;
    DataSource "FIELD COMPANY->ZIP",;
    Name "ZipList"

```

See Also

DEFINE, ID

NextCol()

Returns the number of the next highest column position where an object can be placed in a form.

Property of class

FORM

Description

Use NextCol() in combination with NextRow() to prevent the overlapping of objects in a form. For example, if you create an entry field, NextCol() returns the coordinate of the next available column and NextRow() returns the coordinate of the next available row. You can place a new object at these coordinates without covering the entry field.

The column coordinates returned by NextCol() are as wide as the average width of characters in the active font of the form.

Example

```

LOCAL f
f=NEW EntryForm()
f.OPEN()
CLASS EntryForm of FORM
DEFINE ENTRYFIELD Fname OF this;
    PROPERTY;
    Top 3, Left 2, Width 20,;
    Value SPACE(20)
DEFINE ENTRYFIELD Lname OF this;
    PROPERTY;
    Top 5, Left 2, Width 20,;
    Value SPACE(20)
    * Use NextRow() and NextCol() to define a
    * pushbutton adjacent to and 4 columns to
    * the right of the entry field Lname.
DEFINE PUSHBUTTON PB1 OF this;
    PROPERTY Text "Proceed",;
    Top this.NextRow()-2,;
    Left this.NextCol()+4,;
    Width 10
ENDCLASS

```

N

NextIndex()

See Also

COL(), NextRow(), ROW(), ScaleFontName, ScaleFontSize

NextIndex()

Returns the subscript of the next element in an associated array.

Property of class

ASSOCARRAY

Description

Use NextIndex() to step through the elements in an associated array object, starting from the first element in the array. Generally, you'll use FirstIndex to position yourself on the first element in the array, and then use NextIndex() to step through the elements in order. NextIndex() requires one parameter: the index of the element of the array to start from when looking for the next element.

Example

See FirstIndex for an example of NextIndex().

See Also

FirstIndex, IsIndex()

NextObj

Contains a reference to the object that follows the current object in the tabbing order of a parent form.

Property of class

FORM

Data type

Object reference

Description

Use NextObj to reference the next object to receive input focus when the user presses *Tab*.

Use NextObj in Valid subroutines to determine if validation is needed before moving to the next object. For example, the following commands determine if the next object is a Cancel pushbutton:

```
IF (MyForm.NextObj.Name = "Cancel")
    RETURN .T.    && No validation is required.
ELSE
```

```

RETURN Validate()  && Validation routine
ENDIF

```

Example

The following example uses `ActiveControl` and `NextObj` to return the name of the current and next entry field objects when the right mouse is clicked on a form. This would be useful on a form where entry fields had no accompanying text to identify the fields:

```

LOCAL f
f=NEW EntryForm()
f.OPEN()
CLASS Entryform OF FORM
  this.View="Company.DBF"
  this.OnRightMouseDown={;Fcs="Current Field: " + ;
    Form.ActiveControl.Name; Form.Txt1.Text=Fcs}
  this.OnRightMouseUp={;Fcs2="Next Field:      " + ;
    Form.NextObj.Name; Form.Txt2.Text=Fcs2}
DEFINE ENTRYFIELD Company OF THIS;
  PROPERTY Datalink "Company->Company",;
  Top 3, Left 1, Width 20, Name "Company"
DEFINE ENTRYFIELD State OF THIS;
  PROPERTY Datalink "Company->State_Prov",;
  Top 5, Left 1, Name "State"
DEFINE TEXT Txt1 OF THIS;
  PROPERTY Text " ",;
  Top 8, Left 1, Width 25
DEFINE TEXT Txt2 OF THIS;
  PROPERTY Text " ",;
  Top 9, Left 1, Width 25
ENDCLASS

```

N

See Also

`ActiveControl`, `_curobj`, `CUATab`, `First`, `SetFocus()`

NextRow()

Returns the number of the next highest row position where an object can be placed in a form.

Property of class

Form

Description

Use `NextRow()` in combination with `NextCol()` to prevent the overlapping of objects in a form. For example, if you create an entry field, `NextRow()` returns the coordinate of the next available row, and `NextCol()` returns the coordinate of the next available column; you can place a new object at these coordinates without covering the entry field.

The row coordinates returned by NextRow() are as wide as the average line of characters in the active font of the form.

Example

See NextCol() for an example of using NextRow().

See Also

COL(), NextRow(), ROW()

Notify()

Notifies a client application that a dBASE item was changed.

Property of class

DDETOPIC

Description

Use Notify() in a DDE server program to tell a client application that an item in the dBASE server session was changed.

OnPoke subroutines often execute Notify() when an external application sends dBASE a Poke request. For example, a Quattro Pro data-exchange application might use its {POKE} command to send dBASE a value, causing the OnPoke subroutine to execute. The OnPoke subroutine could insert the value into a field, then execute Notify() to inform Quattro Pro that the field changed.

Notify() accepts one parameter, *<item>*. Use this parameter to tell the client application which field, variable, or array element changed. For example, if an OnPoke subroutine changed the value in a field named LASTNAME, the subroutine might execute the following command:

```
MyDDETopic.Notify("LASTNAME")
```

The client application must establish a hot link before Notify() has an effect. For information on hot links, see OnAdvise().

Example

See CLASS DDETOPIC for an example of using Notify().

See Also

Advise(), Execute(), Initiate(), OnNewValue, Peek(), Poke(), Server, Terminate(), TimeOut, Topic, Unadvise()

OldStyle

Determines if a check box or a radio button is displayed in the default Windows style or in 3-D style.

Property of class

CHECKBOX, ENTRYFIELD, LISTBOX, RADIOBUTTON, RECTANGLE, SPINBOX, TEXT

Data type

Logical

Default

The default for OldStyle is false (.F.).

Description

Use OldStyle to determine the appearance of an object. When you set OldStyle to true (.T.), the object is displayed in Windows style; when you set OldStyle to false, the object is displayed in 3-D style.

Example

NEW operator syntax:

```
CheckOld = NEW CHECKBOX(this)
CheckOld.Oldstyle = .T.
```

DEFINE object syntax:

```
DEFINE CHECKBOX CheckOld OF THIS;
    Property;
    Oldstyle .T.
```

See Also

PatternStyle, BorderStyle

OleType

Returns a number that reveals whether an OLE field is empty, contains an embedded document, or contains a link to a document file.

Property of class

OLE

Data type

Character

Default

The default for OleType is 0 (empty).

Description

Use OleType to determine whether an OLE field is empty, contains a link to a document file, or contains an embedded document.

An OLE viewer displays an OLE document (sometimes called an OLE object). An OLE document can be a graphic image, a document created by a word processor, or any other data object created by an external application. This external application is known as the *OLE server*. (The OLE server must have OLE capability.)

For example, a graphic image created in Paintbrush can be an OLE document, and Paintbrush can be an OLE server, if you do one of the following:

- Embed the document (or a portion of it) in an OLE field with the Cut and Paste commands of the Edit menu.
- Link the file containing the document to an OLE field with the Cut and Paste Link commands of the Edit menu.

When the current OLE field is empty—that is, when it contains no embedded OLE document and no link to an OLE document—OleType contains 0. When the OLE field contains an embedded document, OleType contains 1. When the OLE field contains a link to a document file, OleType contains 2.

OleType is a read-only property.

Example

See DoVerb() for an example of using OleType.

See Also

LinkFileName, ServerName

OnAdvise

Executes a subroutine when an external application requests a DDE hot link to an item in a dBASE server topic.

Property of class

DDETOPIC

Data type

Function pointer or codeblock

Description

Use OnAdvise in a DDE server program to respond to a request for a hot link and to determine which dBASE data item the link applies to. (A hot link lets you use the Notify() method to tell the client when the item changes.)

OnAdvise receives *<item>*, a parameter that identifies the dBASE data item. Your server program can use this item for any purpose. For example, it might identify a dBASE field, variable, or array element.

Item names are often held in tables or array objects when multiple hot links are established. For example, a client application might request a hot link to a field, passing the field name through the *<item>* parameter. Each time, the OnAdvise subroutine could place the field name in an array object.

The OnPoke subroutine could search this array object each time a field is changed; if the name of the changed field is found in the array object, the routine could execute Notify().

Example

See CLASS DDETOPIC for an example of using OnAdvise property.

See Also

Advise(), Execute(), Initiate(), OnNewValue, Peek(), Poke(), Server, Terminate(), TimeOut, Topic, Unadvise()

OnAppend

Executes a subroutine when a record is added to a table.

Property of class

BROWSE, FORM

Data type

Function pointer or codeblock

O

Description

Use OnAppend to perform an action each time a new record is created.

OnAppend lets your application respond each time new information is added to a table. For example, a browse object might use an OnAppend subroutine to set a logical field named NEWFIELD to true (.T.) automatically each time the user adds a new record.

Example

NEW operator syntax:

```
Browser = NEW BROWSE(this)
Browser.text = "Sample for OnAppend"
Browser.OnAppend = DataCheck
```

DEFINE object syntax:

```
DEFINE BROWSE Browser OF THIS;
PROPERTY;
Text "Sample for OnAppend",;
```

```

    OnAppend DataCheck
PROCEDURE DataCheck
IF LEN(Client->Name) = 0
    ? "You must add a name"
ENDIF

```

See Also

OnChange, OnNavigate

OnChange

Executes a subroutine when the user changes the value displayed in an object.

Property of class

BROWSE, CHECKBOX, COMBOBOX, EDITOR, ENTRYFIELD, FORM, OLE, RADIOBUTTON, SCROLLBAR, SPINBOX

Data type

Function pointer or codeblock

Description

Use OnChange to execute a routine when the user performs any of the following actions:

- Changes a value in a field and moves to another row in a browse object
- Inserts or removes a checkmark in a check box
- Changes a value in an entry field
- Changes a value in the text box portion of a combo box or a spin box
- Selects a different radio button
- Moves the slider button in a scrollbar object

The OnChange subroutine of an OLE object executes each time the record pointer moves from one record to another.

For example, an application might let the user change a customer's social security number through an entry field. To verify the change, you can assign to the OnChange property a function that displays a dialog box prompting the user to confirm the change.

Like other event properties, the OnChange property accepts a

- Function
- Procedure
- Codeblock

For information on these subroutines, see Chapter 4 in the *Programmer's Guide*. For information on writing event-handling routines, see Chapter 14 in the *Programmer's Guide*.

Note You can write a subroutine with the Procedure Editor, a window in which you enter dBASE program code. To access the Procedure Editor, click on the Tool button next to the OnChange item in the Inspector.

Example

NEW operator syntax:

```
Company = NEW EntryField(this)
Company.Top = 3
Company.Left = 1
Company.DataLink = "Company->Company"
Company.OnChange = ChkChg
```

DEFINE object syntax:

```
DEFINE EntryField Company OF THIS;
PROPERTY Top 3, Left 1, ;
DataLink "Company->Company", ;
OnChange ChkChg
```

See Also

OnAppend, OnNavigate

OnChar

Executes a subroutine when a “printable” key or key combination is pressed while the control has focus.

Property of class

PAINTBOX

Data type

Function pointer or codeblock

Description

If you have created a paintbox object to develop a custom edit control, use OnChar to determine actions that should take place when the object has focus and the user presses a key or key combination that can be printed. (To specify actions triggered by other keys, see OnKeyDown.)

OnChar is similar to OnKeyDown. However, OnChar returns nothing for non-printable keys, such as Shift or CapsLock, while OnKeyDown returns a value for any key pressed.

Three numeric parameters are passed to the OnChar event:

- *<nChar>*: the scan code of the key or key combination
- *<nReptCnt>*: the number of times the keystroke is repeated based on how long the key is held down
- *<nFlags>*: a parameter used to specify if the Shift or Ctrl key was pressed

(For more information on nFlags, see any of the On Mouse events, such as OnLeftMouseDown.)

Example

```

local f
f = new SAMPLEFORM()
f.Open()
CLASS SAMPLEFORM OF FORM
  this.TopMost = .F.
  this.Height = 8.7051
  this.Text = "Sample"
  this.Left = 47.166
  this.Top = 3.2344
  this.PageNo = 1
  this.Width = 27
  this.ColorNormal = "N/BTNFACE"
  DEFINE RECTANGLE RECTANGLE1 OF THIS;
    PROPERTY;
      Height 5.5;;
      Text " ";
      Left 4.5;;
      Border .T.;;
      Top 1.5;;
      PageNo 1;;
      Width 19;;
      ColorNormal "BTNTEXT/BTNFACE"
  DEFINE PAINTBOX PAINTBOX1 OF THIS;
    PROPERTY;
      OnKeyDown CLASS::PAINTBOX1_ONKEYDOWN;;
      OnChar CLASS::PAINTBOX1_ONCHAR;;
      Height 5;;
      OnPaint CLASS::PAINTBOX1_ONPAINT;;
      Left 6;;
      Top 2;;
      PageNo 1;;
      OnFormSize CLASS::PAINTBOX1_ONFORMSIZE;;
      Width 17;;
      OnKeyUp CLASS::PAINTBOX1_ONKEYUP;;
      ColorNormal "BTNTEXT/BTNFACE"
  Procedure PAINTBOX1_OnChar(nChar, nRepCnt, nFlags)
    ? "OnChar values:"
    ? nChar
    ? nRepCnt
    ? nFlags
    ?
  RETURN

  Procedure PAINTBOX1_OnFormSize(sizeType, width, height)
    ? "OnFormSize values:"
    ? sizeType
    ? width
    ? height
    ?
  RETURN

```

```

Procedure PAINTBOX1_OnKeyDown(nChar, nRepCnt, nFlags)
    ? "OnKeyDown values:"
    ? nChar
    ? nRepCnt
    ? nFlags
    ?
RETURN

Procedure PAINTBOX1_OnKeyUp(nChar, nRepCnt, nFlags)
    ? "OnKeyUp values:"
    ? nChar
    ? nRepCnt
    ? nFlags
    ?
Return

Procedure PAINTBOX1_OnPaint
    ? "PaintBox painted!"
    ?
RETURN

ENDCLASS

```

See Also

Key, OnKeyDown, OnKeyUp, OnLeftMouseDown

OnClick

Executes a subroutine when the user chooses a pushbutton or a menu item.

Property of class

MENU, PUSHBUTTON

Data type

Function pointer or codeblock

Description

Use OnClick to assign an action to a pushbutton or a menu item. A Quit pushbutton, for example, might have a function assigned to its OnClick property that terminates program execution.

Like other event properties, the OnClick property accepts a

- Function
- Procedure
- Codeblock

For information on these subroutines, see Chapter 4 in the *Programmer's Guide*. For information on writing event-handling routines, see Chapter 14 in the *Programmer's Guide*.

Note You can write a subroutine with the Procedure Editor, a window in which you enter dBASE program code. To access the Procedure Editor, click on the Tool button next to the OnClick item in the Inspector.

The subroutine specified by OnClick is executed only if

- the Enabled property of the pushbutton is true
- the When property of the pushbutton evaluates to true (.T.)

Example

NEW operator syntax:

```
Next = NEW PUSHBUTTON(this)
Next.Text = "Next Entry"
Next.OnClick = {;SKIP}
```

DEFINE object syntax:

```
DEFINE PUSHBUTTON Next OF THIS;
Property;
Text "Next Entry",;
Onclick {;SKIP}
```

See Also

OnLeftDbClick, OnLeftMouseDown, OnLeftMouseUp, OnMiddleDbClick, OnMiddleMouseDown, OnMiddleMouseUp, OnRightDbClick, OnRightMouseDown, OnRightMouseUp, OnSelection

OnClose

Executes a subroutine when a form is closed.

Property of class

FORM, OLE

Data type

Function pointer or codeblock

Description

Use OnClose to perform an action automatically when a form is closed. For example, the OnClose property might perform clean-up operations such as closing procedure files, restoring settings, closing tables, and releasing objects.

Like other event properties, the OnClose property accepts a

- Function

- Procedure
- Codeblock

For information on these subroutines, see Chapter 4 in the *Programmer's Guide*. For information on writing event-handling routines, see Chapter 14 in the *Programmer's Guide*.

Before executing the OnClose subroutine, dBASE does the following:

- 1 Executes the Valid subroutine (if any) of the object that currently has input focus. If it returns a value of false (.F.), the form does not close.
- 2 Executes the OnLostFocus subroutine (if any) of the object that currently has input focus.
- 3 Executes the OnLostFocus subroutine (if any) of the form.

After dBASE executes the OnClose subroutine, it closes the form, its objects, and all its child forms.

The OnClose subroutine of an OLE object executes when the parent form is closed.

Note You can write a subroutine with the Procedure Editor, a window in which you enter dBASE program code. To access the Procedure Editor, click on the Tool button next to the OnClose item in the Inspector.

Example

NEW operator syntax:

```
LOCAL f
f=NEW EntryForm()
f.OPEN()
CLASS EntryForm OF FORM
    this.OnClose Cleanup  && Function that closes;
                        tables and other program;
                        elements
ENDCLASS
```

DEFINE object syntax:

```
DEFINE FORM EntryForm;
    PROPERTY OnClose Cleanup
```

See Also

OnChange, OnClick, OnGotFocus, OnHelp, OnLostFocus, OnMove, OnOpen, OnSize

OnExecute

Executes a subroutine when a client application sends a command string to a DDE server program.

Property of class

DDETOPIC

Data type

Function pointer or codeblock

Description

Use the OnExecute property to perform an action when an external application (known as the *client application*) sends a directive to *Visual* dBASE. This directive can be any string of characters.

OnExecute receives `<cmd>`, a parameter that contains the directive sent by the client application. If the directive is a dBASE command, the subroutine can execute it. For example, the client application might send the character string "CLOSE DATABASES". The subroutine could execute the command using the macro substitution function, as with

```
&Cmd
```

In this case, the macro substitution function (&) makes dBASE treat the contents of *Cmd* as a command instead of a character string.

If the directive is not a command, the subroutine can use it to make branching decisions. For example, a stock trading routine might receive either of two character strings, "BUY" or "SELL". The routine could use an IF...ELSE...ENDIF statement to execute one procedure or another accordingly.

For more information on the DDETopic object class, see CLASS DDETOPIC and Chapter 26 in the *Programmer's Guide*.

Example

See CLASS DDETOPIC for an example of using OnExecute property.

See Also

Advise(), Execute(), Initiate(), OnNewValue, OnPeek, OnPoke, Peek(), Poke(), Server, Terminate(), TimeOut, Topic, Unadvise()

OnFormSize

Executes a subroutine whenever the parent form of a paintbox object is resized.

Property of class

PAINTBOX

Data type

Function pointer or codeblock

Description

OnFormSize is called whenever the parent form of a paintbox object is resized, restored, or maximized. This lets you reposition or resize the object based on the form's new size. For example, you could use OnFormSize to implement behavior similar to the Anchor

property of the TABBOX class. keeping the bottom of the paintbox object positioned near the bottom of the form.

OnFormSize is similar to OnPaint. However, OnPaint is triggered when the parent form is opened and when items covering the paintbox object are moved away, while OnFormSize is not.

Example

See OnChar for an example.

OnGotFocus

Executes a subroutine when an object receives focus.

Property of class

BROWSE, CHECKBOX, COMBOBOX, EDITOR, ENTRYFIELD, FORM, LISTBOX, OLE, PUSHBUTTON, RADIOBUTTON, SCROLLBAR, SPINBOX, TABBOX

Data type

Function pointer or codeblock

Description

Use OnGotFocus to perform an action automatically each time an object is selected. For example, a browse object that accesses sensitive information might use its OnGotFocus property to display a dialog box that prompts for a password. If the user fails to enter the correct password, focus could be given to another object.

Like other event properties, the OnGotFocus property accepts a

- Function
- Procedure
- Codeblock

For information on these subroutines, see Chapter 4 in the *Programmer's Guide*. For information on writing event-handling routines, see Chapter 14 in the *Programmer's Guide*.

Note You can write a subroutine with the Procedure Editor, a window in which you enter dBASE program code. To access the Procedure Editor, click on the Tool button next to the OnGotFocus item in the Inspector.

OnGotFocus differs from When, which specifies a condition that must evaluate to true (.T.) before an object can receive focus.

Example

NEW operator syntax:

```
Company = NEW ENTRYFIELD(this)
Company.Top = 3
Company.Left = 1
```

```
Company.Datalink = "Company"
Company.OnGotFocus = GetReady
```

DEFINE object syntax:

```
DEFINE Entryfield Company OF THIS;
  Property;
  Top 3, Left 1,;
  Datalink "Company",;
  OnGotFocus GetReady
```

See Also

OnClick, OnClose, OnHelp, OnLostFocus, OnMove, OnOpen, OnSize

OnHelp

Executes a subroutine when the user presses *F1* while an object has focus.

Property of class

BROWSE, CHECKBOX, COMBOBOX, EDITOR, ENTRYFIELD, FORM, LISTBOX, MENU, PUSHBUTTON, RADIOBUTTON, SCROLLBAR, SPINBOX, TABBOX

Data type

Function pointer or codeblock

Description

Use OnHelp to execute a dBASE routine (instead of a Help topic in a Windows Help file) when the user presses *F1*. For example, a Human Resources program might allow users to search for employee names in a combo box. The OnHelp property could open a dialog box explaining how to find names quickly by entering characters instead of scrolling the combo box prompts.

Like other event properties, the OnChange property accepts

- Functions
- Procedures
- Codeblocks

For information on these subroutines, see Chapter 4 in the *Programmer's Guide*. For information on writing event-handling routines, see Chapter 14 in the *Programmer's Guide*.

Note You can write a subroutine with the Procedure Editor, a window in which you enter dBASE program code. To access the Procedure Editor, click on the Tool button next to the OnHelp item in the Inspector.

The subroutine you specify with OnHelp overrides the compiled Help specified in HelpID and HelpFile.

Example

NEW operator syntax:

```
Resume = New EDITOR(this)
Resume.Top = 3
Resume.Left = 1
Resume.OnHelp = EditHelp
```

DEFINE object syntax:

```
DEFINE EDITOR Resume OF THIS;
  AT 3,1;
  PROPERTY;
  OnHelp EditHelp
```

See Also

HELP, HelpFile, OnChange, OnClick, OnClose, OnGotFocus, OnLostFocus, OnMove, OnOpen, OnSize

OnInitMenu

Specifies code that executes when a menubar or popup is opened.

Property of class

MENUBAR, POPUP

Data type

Function pointer or codeblock

Description

OnInitMenu is called whenever a menubar or popup is invoked, and is processed before the menubar's child menus or the popup is displayed.

You can use OnInitMenu to determine the status of menu items that will be displayed. For example, use OnInitMenu to determine if the Enabled or Checked property of a menu item should be true or false.

Example

```
Parameter FormObj
NEW SAMPLEMENU(FormObj,"Root")
CLASS SAMPLEMENU(FormObj,Name) OF MENUBAR(FormObj,Name)
  this.OnInitMenu = {; ? "Menu opened!"}
  DEFINE MENU FILE OF THIS;
  PROPERTY;
  Text "&File"
  DEFINE MENU EXIT OF THIS.FILE;
  PROPERTY;
  Text "E&xit",;
```

```
OnClick {; Form.Close()}  
ENDCLASS
```

See Also

Checked, Enabled

OnKeyDown

Executes a subroutine when any key or key combination is pressed while the control has focus.

Property of class

PAINTBOX

Data type

Function pointer or codeblock

Description

If you have created a paintbox object to develop a custom edit control, use `OnKeyDown` to determine actions that should take place when the object has focus and the user presses any key or key combination. (To specify actions triggered by printable keys, see `OnChar`.)

`OnChar` is similar to `OnKeyDown`. However, `OnChar` returns nothing for non-printable keys, such as Shift or CapsLock, while `OnKeyDown` returns a value for any key pressed.

Three numeric parameters are passed to the `OnKeyDown` event:

- *nChar* - the scan code of the key or key combination
- *nReptCnt* - the number of times the keystroke is repeated based on how long the key is held down
- *nFlags* - a parameter used to specify if the Shift or Ctrl key was pressed

(For more information on *nFlags*, see any of the On Mouse events, such as `OnLeftMouseDown`.)

Example

See `OnChar` for an example.

See AlsoKey, `OnKeyDown`, `OnKeyUp`, `OnLeftMouseDown`

OnKeyUp

Executes a subroutine when any key or key combination is released while the control has focus.

Property of class

PAINTBOX

Data type

Function pointer or codeblock

Description

If you have created a paintbox object to develop a custom edit control, use OnKeyUp to determine actions that should take place when the object has focus and the user releases a key. (To specify actions triggered when the user presses a key, see OnKeyDown.)

Three numeric parameters are passed to the OnKeyUp event:

- *nChar* - the scan code of the key or key combination
- *nReptCnt* - the number of times the keystroke is repeated based on how long the key is held down
- *nFlags* - a parameter used to specify if the Shift or Ctrl key was pressed

(For more information on nFlags, see any of the On Mouse events, such as OnLeftMouseUp.)

Example

See OnChar for an example

See Also

OnChar, OnKeyDown, OnLeftMouseUp



OnLeftDoubleClick

Executes a subroutine when the user double-clicks a form or an object.

Property of class

BROWSE, CHECKBOX, COMBOBOX, EDITOR, ENTRYFIELD, FORM, IMAGE, LISTBOX, MENU, PUSHBUTTON, RADIOBUTTON, RECTANGLE, SCROLLBAR, SPINBOX, TABBOX, TEXT

Data type

Function pointer or codeblock

Description

Use OnLeftDbIcIck to perform an action when the user double-clicks with the left mouse button. OnLeftDbIcIck can also trap *Shift*, *Ctrl*, middle mouse button, or right mouse button presses if they occur at the same time the user double-clicks the button.

OnLeftDbIcIck passes three parameters to its subroutine:

- *<Flags>* is a single-byte value that tells you which other keys and mouse buttons were pressed when the user double-clicked the button.
- *<Col>* is the horizontal position of the mouse when the user double-clicked the button.
- *<Row>* is the vertical position of the mouse when the user double-clicked the button.

<Flags> parameter If you want, you can use the *Flags* parameter with the BITSET() function to detect different key and mouse button combinations. BITSET() evaluates a numeric value and returns true (.T.) or false (.F.), depending on whether a particular bit in the value is on (1) or off (0). You give BITSET() two parameters:

- The *<Flags>* value itself
- The bit in *<Flags>* to evaluate

The values you can pass to BITSET() through the second parameter are as follows:

Simultaneous keypress	Second parameter
Right mouse button	1
<i>Shift</i>	2
<i>Ctrl</i>	3
Middle mouse button	4

For example, to determine if the user pressed *Shift* while double-clicking the object, use BITSET(Flags, 2). If the user did, BITSET() returns true. If BITSET(Flags, 3) and BITSET(Flags, 1) both return true, *Ctrl* and right mouse button were pressed during the event. You can use such information in IF...ELSE...ENDIF or DO CASE...ENDCASE statements to assign different actions to different key and mouse button combinations.

<Col> and <Row> parameters The *<Col>* and *<Row>* parameters indicate where the double-click event occurred, letting you make decisions accordingly. For example, the OnLeftDbIcIck property of a form might execute different actions, depending on where the double-click event occurred in the form. The *<Col>* and *<Row>* coordinates are expressed in character units.

Like other event properties, the OnLeftDbIcIck property accepts:

- Functions
- Procedures
- Codeblocks

For information on these subroutines, see Chapter 4 in the *Programmer's Guide*. For information on writing event-handling subroutines, see Chapter 14 in the *Programmer's Guide*.

Note You can write a subroutine with the Procedure Editor, a window in which you enter dBASE program code. To access the Procedure Editor, click the Tool button next to the OnLeftDbClick item in the Inspector.

Example

NEW operator syntax:

```
LOCAL f
f=NEW EntryForm
f.OPEN()
CLASS EntryForm OF FORM
    this.OnLeftDbClick = SelectEntry
ENDCLASS
```

DEFINE object syntax:

```
DEFINE FORM EntryForm;
    Property OnLeftDbClick SelectEntry
```

See Also

BITSET(), OnLeftMouseDown, OnLeftMouseUp, OnMiddleDbClick, OnMiddleMouseDown, OnMiddleMouseUp, OnRightDbClick, OnRightMouseDown, OnRightMouseUp

OnLeftMouseDown

Executes a subroutine when the user presses the left mouse button while the pointer is over a form or an object.

Property of class

BROWSE, CHECKBOX, COMBOBOX, EDITOR, ENTRYFIELD, FORM, IMAGE, LISTBOX, MENU, PUSHBUTTON, RADIOBUTTON, RECTANGLE, SCROLLBAR, SPINBOX, TABBOX, TEXT

Data type

Function pointer or codeblock

Description

Use OnLeftMouseDown to perform an action when the user presses the left mouse button. OnLeftMouseDown can also trap *Shift*, *Ctrl*, middle mouse button, or right mouse button presses if they occur at the same time the user presses the button.

OnLeftMouseDown passes three parameters to its subroutine:

- *<Flags>* is a single-byte value that tells you which other keys and mouse buttons were pressed when the user pressed the button.
- *<Col>* is the horizontal position of the mouse when the user pressed the button.
- *<Row>* is the vertical position of the mouse when the user pressed the button.

<Flags> parameter If you want, you can use the *Flags* parameter with the BITSET() function to detect different key and mouse button combinations. BITSET() evaluates a numeric value and returns true (.T.) or false (.F.), depending on whether a particular bit in the value is on (1) or off (0). You give BITSET() two parameters:

- The <Flags> value itself
- The bit in <Flags> to evaluate

The values you can pass to BITSET() through the second parameter are as follows:

Simultaneous keypress	Second parameter
Right mouse button	1
Shift	2
Ctrl	3
Middle mouse button	4

For example, to determine if the user pressed *Shift* while clicking the object, use BITSET(Flags, 2). If the user did, BITSET() returns true. If BITSET(Flags, 3) and BITSET(Flags, 1) both return true, *Ctrl* and the right mouse button were pressed during the event. You can use such information in IF...ELSE...ENDIF or DO CASE...ENDCASE statements to assign different actions to different key and mouse button combinations.

<Col> and <Row> parameters The <Col> and <Row> parameters indicate where the click event occurred, letting you make decisions accordingly. For example, the OnLeftMouseDown property of a form might execute different actions, depending on where the click event occurred in the form. The <Col> and <Row> coordinates are expressed in character units.

Like other event properties, the OnLeftMouseDown property accepts:

- Functions
- Procedures
- Codeblocks

For information on these subroutines, see Chapter 4 in the *Programmer's Guide*. For information on writing event-handling routines, see Chapter 14 in the *Programmer's Guide*.

Note You can write a subroutine with the Procedure Editor, a window in which you enter dBASE program code. To access the Procedure Editor, click the Tool button next to the OnLeftMouseDown item in the Inspector.

Example

NEW operator syntax:

```
LOCAL f
f=NEW EntryForm()
f.OPEN()
CLASS EntryForm OF FORM
  this.OnLeftMouseDown = OkToMove
  this.OnLeftMouseUp = StopMove
ENDCLASS
```

DEFINE object syntax:

```
DEFINE FORM EntryForm;
  PROPERTY OnLeftMouseDown OkToMove,;
    OnLeftMouseUp StopMove
OPEN FORM EntryForm
```

See Also

BITSET(), OnLeftDbClick, OnLeftMouseUp, OnMiddleDbClick, OnMiddleMouseDown, OnMiddleMouseUp, OnRightDbClick, OnRightMouseDown, OnRightMouseUp

OnLeftMouseUp

Executes a subroutine when the user releases the left mouse button while the pointer is over a form or an object.

Property of class

BROWSE, CHECKBOX, COMBOBOX, EDITOR, ENTRYFIELD, FORM, IMAGE, LISTBOX, MENU, PUSHBUTTON, RADIOBUTTON, RECTANGLE, SCROLLBAR, SPINBOX, TABBOX, TEXT

Data type

Function pointer or codeblock

Description

Use OnLeftMouseUp to perform an action when the user releases the left mouse button. OnLeftMouseUp can also trap *Shift*, *Ctrl*, middle mouse button, or right mouse button presses if they occur at the same time the user releases the button.

OnLeftMouseUp passes three parameters to its subroutine:

- *<Flags>* is a single-byte value that tells you which other keys and mouse buttons were pressed when the user released the button.
- *<Col>* is the horizontal position of the mouse when the user released the button.
- *<Row>* is the vertical position of the mouse when the user released the button.

<Flags> parameter If you want, you can use the *Flags* parameter with the BITSET() function to detect different key and mouse button combinations. BITSET() evaluates a numeric value and returns true (.T.) or false (.F.), depending on whether a particular bit in the value is on (1) or off (0). You give BITSET() two parameters:

- The *<Flags>* value itself
- The bit in *<Flags>* to evaluate

0

The values you can pass to BITSET() through the second parameter are as follows:

Simultaneous keypress	Second parameter
Right mouse button	1
Shift key	2
Ctrl key	3
Middle mouse button	4

For example, to determine if the user pressed *Shift* while clicking the object, use BITSET(Flags, 2). If the user did, BITSET() returns true. If BITSET(Flags, 3) and BITSET(Flags, 1) both return true, *Ctrl* and the right mouse button were pressed during the event. You can use such information in IF...ELSE...ENDIF or DO CASE...ENDCASE statements to assign different actions to different key and mouse button combinations.

<Col> and <Row> parameters The <Col> and <Row> parameters indicate where the upward-click event occurred, letting you make decisions accordingly. For example, the OnLeftMouseUp property of a form might execute different actions, depending on where the left upward-click event occurred in the form. The <Col> and <Row> coordinates are expressed in character units.

Like other event properties, the OnLeftMouseUp property accepts:

- Functions
- Procedures
- Codeblocks

For information on these subroutines, see Chapter 4 in the *Programmer's Guide*. For information on writing event-handling subroutines, see Chapter 14 in the *Programmer's Guide*.

Note You can write a subroutine with the Procedure Editor, a window in which you enter dBASE program code. To access the Procedure Editor, click on the Tool button next to the OnLeftMouseUp item in the Inspector.

Example

See OnLeftMouseDown for an example of OnLeftMouseUp.

See Also

BITSET(), OnLeftDbClick, OnLeftMouseDown, OnMiddleDbClick, OnMiddleMouseDown, OnMiddleMouseUp, OnRightDbClick, OnRightMouseDown, OnRightMouseUp

OnLostFocus

Executes a subroutine when focus is removed from an object.

Property of class

BROWSE, CHECKBOX, COMBOBOX, EDITOR, ENTRYFIELD, FORM, LISTBOX, PUSHBUTTON, RADIOBUTTON, SCROLLBAR, SPINBOX, TABBOX

Data type

Function pointer or codeblock

Description

Use OnLostFocus to perform an action automatically when an object loses focus. For example, an entry field might provide access to an important table field in which a valid entry must be made. When the user attempts to move focus to another object, the OnLostFocus property could present a dialog box containing the confirmation prompt "Are you sure?" with two pushbuttons labeled Yes and Edit Again.

Like other event properties, the OnLostFocus property accepts a

- Function
- Procedure
- Codeblock

For information on these subroutines, see Chapter 4 in the *Programmer's Guide*. For information on writing event-handling routines, see Chapter 14 in the *Programmer's Guide*.

Note You can write a subroutine with the Procedure Editor, a window in which you enter dBASE program code. To access the Procedure Editor, click on the Tool button next to the OnLostFocus item in the Inspector.

OnLostFocus differs from Valid, which specifies a condition that must evaluate to true (.T.) before the object can lose focus.

Example

NEW operator syntax:

```
Spin1 = NEW SPINBOX(this)
Spin1.Datalink = "TaxRate"
Spin1.Top = 2
Spin1.Left = 4
Spin1.Height = 2
Spin1.OnLostFocus = Recompute
* Recompute is a FUNCTION that performs an
* action and returns .T. or some value.
```



DEFINE object syntax:

```
DEFINE SPINBOX Spin1 OF THIS;
  PROPERTY Datalink "Taxrate",;
  Top 2, Left 4, Height 2,;
  OnLostFocus Recompute
```

See Also

OnChange, OnClick, OnClose, OnGotFocus, OnHelp, OnMove, OnOpen, OnSize

OnMiddleDblClick

Executes a subroutine when the user double-clicks with the middle mouse button while the pointer is on a form or an object.

Property of class

BROWSE, CHECKBOX, COMBOBOX, EDITOR, ENTRYFIELD, FORM, IMAGE, LISTBOX, MENU, PUSHBUTTON, RADIOBUTTON, RECTANGLE, SCROLLBAR, SPINBOX, TABBOX, TEXT

Data type

Function pointer or codeblock

Description

Use OnMiddleDblClick to perform an action when the user double-clicks with the middle mouse button. OnMiddleDblClick can also trap *Shift*, *Ctrl*, left mouse button, or right mouse button presses if they occur at the same time the user double-clicks the button.

OnMiddleDblClick passes three parameters to its subroutine:

- *<Flags>* is a single-byte value that tells you which other keys and mouse buttons were pressed when the user double-clicked the button.
- *<Col>* is the horizontal position of the mouse when the user double-clicked the button.
- *<Row>* is the vertical position of the mouse when the user double-clicked the button.

<Flags> parameter If you want, you can use the *Flags* parameter with the BITSET() function to detect different key and mouse button combinations. BITSET() evaluates a numeric value and returns true (.T.) or false (.F.), depending on whether a particular bit in the value is on (1) or off (0). You give BITSET() two parameters:

- The *<Flags>* value itself
- The bit in *<Flags>* to evaluate

The values you can pass to BITSET() through the second parameter are as follows:

Simultaneous keypress	Second parameter
Left mouse button	0
Right mouse button	1
<i>Shift</i>	2
<i>Ctrl</i>	3

For example, to determine if the user pressed *Shift* while clicking the object, use BITSET(Flags, 2). If the user did, BITSET() returns true. If BITSET(Flags, 3) and BITSET(Flags, 0) both return true, *Ctrl* and the left mouse button were pressed during the event. You can use such information in IF...ELSE...ENDIF or DO CASE...ENDCASE statements to assign different actions to different key and mouse button combinations.

<Col> and <Row> parameters The <Col> and <Row> parameters indicate where the middle double-click event occurred, letting you make decisions accordingly. For example, the OnMiddleDbIcClick property of a form might execute different actions, depending on where the double-click event occurred in the form. The <Col> and <Row> coordinates are expressed in character units.

Like other event properties, the OnMiddleDbIcClick property accepts

- Functions
- Procedures
- Codeblocks

For information on these subroutines, see Chapter 4 in the *Programmer's Guide*. For information on writing event-handling subroutines, see Chapter 14 in the *Programmer's Guide*.

Note You can write a subroutine with the Procedure Editor, a window in which you enter dBASE program code. To access the Procedure Editor, click on the Tool button next to the OnMiddleDbIcClick item in the Inspector.

Example

NEW operator syntax:

```
LOCAL F1
F1=NEW Trips()
F1.OPEN()
CLASS Trips OF FORM
    this.Text = "Trip Schedule"
    this.Top = 2
    this.Left = 2
    this.Width = 38
    this.Height = 18
    this.OnMiddleDbIcClick = AnimalKingdom
* AnimalKingdom is a FUNCTION that performs
* an action and returns .T. or some value.
ENDCLASS
```

DEFINE object syntax:

```
DEFINE FORM Trips;
  PROPERTY Text "Trip Schedule",;
  Top 2, Left 2, Width 38, Height 18
  OnMiddleDblClick AnimalKingdom
OPEN FORM Trips
```

See Also

BITSET(), OnLeftDblClick, OnLeftMouseDown, OnLeftMouseUp, OnMiddleMouseDown, OnMiddleMouseUp, OnRightDblClick, OnRightMouseDown, OnRightMouseUp

OnMiddleMouseDown

Executes a subroutine when the user presses the middle mouse button while the pointer is over a form or an object.

Property of class

BROWSE, CHECKBOX, COMBOBOX, EDITOR, ENTRYFIELD, FORM, IMAGE, LISTBOX, MENU, PUSHBUTTON, RADIOBUTTON, RECTANGLE, SCROLLBAR, SPINBOX, TABBOX, TEXT

Data type

Function pointer or codeblock

Description

Use OnMiddleMouseDown to perform an action when the user presses the middle mouse button. OnMiddleMouseDown can also trap *Shift*, *Ctrl*, left mouse button, or right mouse button presses if they occur at the same time the user presses the button.

OnMiddleMouseDown passes three parameters to its subroutine:

- *<Flags>* is a single-byte value that tells you which other keys and mouse buttons were pressed when the user pressed the button.
- *<Col>* is the horizontal position of the mouse when the user pressed the button.
- *<Row>* is the vertical position of the mouse when the user pressed the button.

***<Flags>* parameter** If you want, you can use the *Flags* parameter with the BITSET() function to detect different key and mouse button combinations. BITSET() evaluates a numeric value and returns true (.T.) or false (.F.), depending on whether a particular bit in the value is on (1) or off (0). You give BITSET() two parameters:

- The *<Flags>* value itself
- The bit in *<Flags>* to evaluate

The values you can pass to BITSET() through the second parameter are as follows:

Simultaneous keypress	Second parameter
Left mouse button	0
Right mouse button	1
<i>Shift</i>	2
<i>Ctrl</i>	3

For example, to determine if the user pressed *Shift* while clicking the object, use BITSET(Flags, 2). If the user did, BITSET() returns true. If BITSET(Flags, 3) and BITSET(Flags, 0) both return true, *Ctrl* and the left mouse button were pressed during the event. You can use such information in IF...ELSE...ENDIF or DO CASE...ENDCASE statements to assign different actions to different key and mouse button combinations.

<Col> and <Row> parameters The <Col> and <Row> parameters indicate where the middle click event occurred, letting you make decisions accordingly. For example, the OnMiddleMouseDown property of a form might execute different actions, depending on where the middle click event occurred in the form. The <Col> and <Row> coordinates are expressed in character units.

Like other event properties, the OnMiddleMouseDown property accepts:

- Functions
- Procedures
- Codeblocks

For information on these subroutines, see Chapter 4 in the *Programmer's Guide*. For information on writing event-handling subroutines, see Chapter 14 in the *Programmer's Guide*.

Note You can write a subroutine with the Procedure Editor, a window in which you enter dBASE program code. To access the Procedure Editor, click on the Tool button next to the OnMiddleMouseDown item in the Inspector.

Example

NEW operator syntax:

```
LOCAL F1
F1=NEW Trips()
F1.OPEN()
CLASS Trips OF FORM
    this.Text = "Trip Schedule"
    this.Top = 2
    this.Left = 2
    this.Width = 38
    this.Height = 18
    this.OnMiddleMouseDown = AnimalKingdom
* AnimalKingdom is a FUNCTION that performs
* an action and returns .T. or some value.
ENDCLASS
```

DEFINE object syntax:

```
DEFINE FORM Trips FROM 2,2 TO 20,40;
  PROPERTY Text "Trip Schedule",;
  OnMiddleMouseDown AnimalKingdom
OPEN FORM Trips
```

See Also

BITSET(), OnLeftDbtClick, OnLeftMouseDown, OnLeftMouseUp, OnMiddleDbtClick, OnMiddleMouseUp, OnRightDbtClick, OnRightMouseDown, OnRightMouseUp

OnMiddleMouseUp

Executes a subroutine when the user releases the middle mouse button while the pointer is over a form or an object.

Property of class

BROWSE, CHECKBOX, COMBOBOX, EDITOR, ENTRYFIELD, FORM, IMAGE, LISTBOX, MENU, PUSHBUTTON, RADIOBUTTON, RECTANGLE, SCROLLBAR, SPINBOX, TABBOX, TEXT

Data type

Function pointer or codeblock

Description

Use OnMiddleMouseUp to perform an action when the user releases the middle mouse button. OnMiddleMouseUp can also trap *Shift*, *Ctrl*, left mouse button, or right mouse button presses if they occur at the same time the user releases the button.

OnMiddleMouseUp passes three parameters to its subroutine:

- *<Flags>* is a single-byte value that tells you which other keys and mouse buttons were pressed when the user released the button.
- *<Col>* is the horizontal position of the mouse when the user released the button.
- *<Row>* is the vertical position of the mouse when the user released the button.

<Flags> parameter If you want, you can use the Flags parameter with the BITSET() function to detect different key and mouse button combinations. BITSET() evaluates a numeric value and returns true (.T.) or false, depending on whether a particular bit in the value is on (1) or off (0). You give BITSET() two parameters:

- The *<Flags>* value itself
- The bit in *<Flags>* to evaluate

The values you can pass to `BITSET()` through the second parameter are as follows:

Simultaneous keypress	Second parameter
Left mouse button	0
Right mouse button	1
<i>Shift</i> key	2
<i>Ctrl</i> key	3

For example, to determine if the user pressed *Shift* while clicking the object, use `BITSET(Flags, 2)`. If the user did, `BITSET()` returns true. If `BITSET(Flags, 3)` and `BITSET(Flags, 0)` both return true, *Ctrl* and the left mouse button were pressed during the event. You can use such information in `IF...ELSE...ENDIF` or `DO CASE...ENDCASE` statements to assign different actions to different key and mouse button combinations.

<Col> and <Row> parameters The `<Col>` and `<Row>` parameters indicate where the middle upward-click event occurred, letting you make decisions accordingly. For example, the `OnMiddleMouseUp` property form might execute different actions, depending on where the middle upward-click event occurred in the form. The `<Col>` and `<Row>` coordinates are expressed in character units.

Like other event properties, the `OnMiddleMouseUp` property accepts:

- Functions
- Procedures
- Codeblocks

For information on these subroutines, see Chapter 4 in the *Programmer's Guide*. For information on writing event-handling subroutines, see Chapter 14 in the *Programmer's Guide*.

Note You can write a subroutine with the Procedure Editor, a window in which you enter dBASE program code. To access the Procedure Editor, click on the Tool button next to the `OnMiddleMouseUp` item in the Inspector.

Example

NEW operator syntax:

```
LOCAL F1
F1=NEW Trips()
F1.OPEN()
CLASS Trips OF FORM
  this.Text = "Trip Schedule"
  this.Top = 2
  this.Left = 2
  this.Width = 38
  this.Height = 18
  this.OnMiddleMouseUp = CostSched
* CostSched is a FUNCTION that performs
* an action and returns .T. or some value.
ENDCLASS
```

DEFINE object syntax:

```
DEFINE FORM Trips;
  PROPERTY Text "Trip Schedule",;
  Top 2, Left 2, Width 38, Height 18,;
  OnMiddleMouseUp CostSched
OPEN FORM Trips
```

See Also

BITSET(), OnLeftDbtClick, OnLeftMouseDown, OnLeftMouseUp, OnMiddleDbtClick, OnMiddleMouseDown, OnRightDbtClick, OnRightMouseDown, OnRightMouseUp

OnMouseMove

Executes a subroutine when the user moves the mouse in a form.

Property of class

BROWSE, CHECKBOX, COMBOBOX, EDITOR, ENTRYFIELD, FORM, IMAGE, LISTBOX, PUSHBUTTON, RADIOBUTTON, RECTANGLE, SCROLLBAR, SPINBOX, TABBOX, TEXT

Data type

Function pointer or codeblock

Description

Use OnMove to perform actions automatically when the user moves the mouse.

Like other event properties, the OnMouseMove property accepts

- Functions
- Procedures
- Codeblocks

For information on these subroutines, see Chapter 4 in the *Programmer's Guide*. For information on writing event-handling routines, see Chapter 14 in the *Programmer's Guide*.

OnMouseMove passes three parameters to its subroutine:

- *<Flags>* is a single-byte value that tells you which keys and mouse buttons were pressed while the user moved the mouse. You can interpret this value with the BITSET() function, which examines individual bits in numeric values.
- *<Col>* is the horizontal position of the mouse after the move.
- *<Row>* is the vertical position of the mouse after the move.

The *<Col>* and *<Row>* parameters reveal the new position, letting you make decisions accordingly. For example, when the user moves the mouse to a restricted region of a form, the OnMouseMove subroutine might use *<Col>* and *<Row>* to detect the action

and display a warning dialog box. The *<Col>* and *<Row>* coordinates are expressed in character units.

For more information on interpreting the *<Flags>* parameter, see the mouse event properties (such as `OnLeftMouseDown` and `OnRightMouseDown`).

Notes You can write a subroutine with the Procedure Editor, a window in which you enter dBASE program code. To access the Procedure Editor, click the Tool button next to the `OnMouseMove` item in the Inspector.

Example

NEW operator syntax:

```
LOCAL F1
F1=NEW Trips()
F1.OPEN()
CLASS Trips OF FORM
  this.Text = "Trip Schedule"
  this.Top = 2
  this.Left = 2
  this.Width = 38
  this.Height = 18
  this.OnMouseMove = CostList
* CostList is a FUNCTION that performs
* an action and returns .T. or some value.
ENDCLASS
```

DEFINE object syntax:

```
DEFINE FORM Trips;
  PROPERTY Text "Trip Schedule",;
  Top 2, Left 2, Width 38, Height 18,;
  OnMouseMove CostList
OPEN FORM Trips
```

See Also

`OnClick`, `OnLeftDbClick`, `OnLeftMouseDown`, `OnLeftMouseUp`, `OnMiddleDbClick`, `OnMiddleMouseDown`, `OnMiddleMouseUp`, `OnRightDbClick`, `OnRightMouseDown`, `OnRightMouseUp`

OnMove

Executes a subroutine after a form is opened and after the user moves the form.

Property of class

FORM, TABBOX

Data type

Function pointer or codeblock

Description

Use OnMove to perform actions automatically when a form is opened or moved.

Like other event properties, the OnMove property accepts

- Functions
- Procedures
- Codeblocks

For information on these subroutines, see Chapter 4 in the *Programmer's Guide*. For information on writing event-handling routines, see Chapter 14 in the *Programmer's Guide*.

OnMove passes two parameters to its subroutine:

- <Left> is the new horizontal position of the upper left corner.
- <Top> is the new vertical position of the upper right corner.

Use the Left and Top parameters to make branching decisions. For example, when the user moves a form over another form, the OnMove subroutine of the moved form might use Left and Top to calculate a new position for the obscured form.

Note You can write a subroutine with the Procedure Editor, a window in which you enter dBASE program code. To access the Procedure Editor, click on the Tool button next to the OnMove item in the Inspector.

Example

NEW operator syntax:

```
LOCAL F1
F1=NEW Trips()
F1.OPEN()
CLASS Trips OF FORM
    this.Text = "Trip Schedule"
    this.Top = 2
    this.Left = 2
    this.Width = 38
    this.Height = 18
    this.OnRightClick = {;Form.Move(10,10,20,10)}
    this.OnMove = ShowCost
* ShowCost is a FUNCTION that performs
* an action and returns .T. or some value.
```

```
ENDCLASS
```

DEFINE object syntax:

```
DEFINE FORM Trips;
  PROPERTY Text "Trip Schedule",;
  Top 2, Left 2, Width 38, Height 18,;
  OnRightClick {;Form.Move(10,10,20,10)},;
  OnMove ShowCost
OPEN FORM Trips
```

See Also

OnChange, OnClick, OnClose, OnGotFocus, OnHelp, OnLeftMouseDown, OnLostFocus, OnOpen, OnSize

OnNavigate

Executes a subroutine when the record pointer in a table is moved.

Property of class

BROWSE, FORM

Data type

Function pointer or codeblock

Description

Use OnNavigate to make your application respond each time the user moves from one record to another.

OnNavigate lets you detect the user's movements through a table. For example, a browse object might use an OnNavigate subroutine to log user activity; each time the user moves from one record to another, the subroutine could store the record number in an array object or in another table.

Example

NEW operator syntax:

```
LOCAL f
f = NEW XFORM()
f.Open()

CLASS XFORM OF FORM
  this.HelpFile = ""
  this.Top = 5.71
  this.Height = 15.82
  this.Text = "Form"
  this.Left = 39.00
  this.Width = 72.60
  this.HelpId = ""
```

```

DEFINE BROWSE BROWSE1 OF THIS;
PROPERTY;
    OnNavigate {;?Compcode+SPACE(5)+Contact},;
    Top          3.00,;
    Height       10.00,;
    FontBold .F.,;
    ColorNormal "N/W",;
    Fields "CompCode, Contact",;
    Left         7.00,;
    Width        59.00

```

```
ENDCLASS
```

DEFINE object syntax:

```

USE Contact
SET PRINTER ON
DEFINE FORM Brws FROM 1,1 TO 15,40
DEFINE BROWSE Br1 OF Brws;
    PROPERTY Alias "Contact",;
        Height 15, Fields "CompCode, Contact",;
        Top 1, Left 1, Width 40, Height 15,;
        OnNavigate {;? Compcode+SPACE(5)+Contact}
    OPEN FORM Brws

```

See Also

OnAppend, OnChange

OnNewValue

Executes a subroutine when a hot-linked item in a DDE server document changes.

Property of class

DDELINK

Data type

Function pointer or codeblock

Description

Use OnNewValue to perform an action when a hot-linked server item is changed. A hot link, which you create with the Advise() method, tells the server to notify dBASE when the item changes.

A server document is a file you open in an external application. For example, a data-exchange application might start a session in Quattro Pro for Windows and open one of its spreadsheet files (the server document). You can establish hot links to one or more cells in the spreadsheet with Advise(), then use OnNewValue to specify a codeblock that executes each time one of the cells is changed.

OnNewValue receives two parameters:

- *<item>* identifies the server item written to. It can specify the hot-linked item, such as a field in a table or a cell in a spreadsheet. For example, "A:H3" specifies cell H3 of Page A in a Quattro Pro spreadsheet file.
- *<value>* is the new value of the hot-linked server item.

The codeblock can use these parameters to evaluate the change and make decisions accordingly.

Example

```
PUBLIC LinkObj
LinkObj = NEW DDELINK()
LinkObj.OnNewValue = Valuehandler      && Codeblock or function pointer
LinkObj.Initiate("QPW", "Demo.WB1")
LinkObj.Advise("A:A1")                 && Notified when cell A:A1 changes
```

See Also

Advise(), Execute(), Initiate(), Peek(), Poke(), Server, Terminate(), TimeOut, Topic, Unadvise()

OnOpen

Executes a subroutine when a form is opened.

Property of class

BROWSE, CHECKBOX, COMBOBOX, EDITOR, ENTRYFIELD, FORM, IMAGE, LINE, LISTBOX, OLE, PUSHBUTTON, RADIOBUTTON, RECTANGLE, SCROLLBAR, SPINBOX, TABBOX, TEXT

Data type

Function pointer or codeblock

Description

Use OnOpen to execute an action when a form is opened.

The OnOpen subroutine of an object executes whenever the parent form is opened. The OnOpen subroutine of a form executes any time the form itself is opened.

Like other event properties, the OnOpen property accepts

- Functions
- Procedures
- Codeblocks

For information on these subroutines, see Chapter 4 in the *Programmer's Guide*. For information on writing event-handling routines, see Chapter 14 in the *Programmer's Guide*.

Notes You can write a subroutine with the Procedure Editor, a window in which you enter dBASE program code. To access the Procedure Editor, click on the Tool button next to the OnOpen item in the Inspector.

Any subroutine you assign to the OnMove property of the form also executes when the form is opened.

Example

NEW operator syntax:

```
LOCAL F1
F1=NEW Trips()
F1.OPEN()
CLASS Trips OF FORM
  this.Text = "Trip Schedule"
  this.Top = 2
  this.Left = 2
  this.Width = 38
  this.Height = 18
  this.OnOpen = FltSched
* FltSched is a FUNCTION that performs
* an action and returns .T. or some value.
ENDCLASS
```

DEFINE object syntax:

```
DEFINE FORM Trips FROM 2,2 TO 20,40;
  PROPERTY Text "Trip Schedule",;
  OnOpen FltSched
OPEN FORM Trips
```

See Also

OnClose, OnGotFocus, OnLostFocus, OnMove, OnSize

OnPaint

Executes a subroutine whenever a paintbox object needs to be redrawn.

Property of class

PAINTBOX

Data type

Function pointer or codeblock

Description

OnPaint is called whenever a paintbox object needs to be redrawn. Events that trigger OnPaint include:

- the parent form is opened
- the parent form is resized

- a minimized parent form is restored or maximized
- a window or object which has been covering the paintbox object is moved away

Example

See OnChar for an example.

See Also

OnFormSize

OnPeek

Executes a subroutine when a client application tries to read an item from a DDE server application.

Property of class

DDETOPIC

Data type

Function pointer or codeblock

Description

Use the OnPeek property to send a value to a client application when the client application makes a Peek request.

OnPeek receives *<item>*, a parameter that identifies the dBASE data item that the client application wants to read.

Use the RETURN command to send the requested item to the client. For example, a Quattro Pro for Windows application might execute a {PEEK} command, sending a field name through *<item>*. The OnPoke subroutine could use RETURN to send the field contents to the client, as with

```
RETURN &Item
```

In this case, the macro substitution function (&) makes dBASE treat the contents of *Item* as a field name instead of a character string, so the contents of the field are sent to the client.

For more information on the DDETopic object class, see CLASS DDETOPIC and Chapter 26 in the *Programmer's Guide*.

Example

See CLASS DDETOPIC for an example of using OnPeek property.

See Also

Advise(), Execute(), Initiate(), OnNewValue, OnPoke, Peek(), Poke(), Server, Terminate(), Timeout, Topic, Unadvise()

OnPoke

Executes a subroutine when a client application attempts to insert a value into a DDE server item.

Property of class

DDETOPIC

Data type

Function pointer or codeblock

Description

Use the OnPoke property to receive a value from a client application and insert it in a data element when the client application makes a Poke request.

OnPoke receives two parameters:

- *<item>* Identifies the data item in which the value is inserted. This item can be any string.
- *<value>* Identifies the value to insert in the data item.

For example, a client application might send a field name and a value to put in the field. The OnPoke subroutine might insert the value in the field using the REPLACE command and the macro substitution function, as with:

```
REPLACE &Item WITH Value
```

In this case, the macro substitution function (&) makes dBASE treat the contents of *Item* as a field name instead of a character string.

For more information on the DDETopic object class, see CLASS DDETOPIC and Chapter 26 in the *Programmer's Guide*.

Note If a client established a hot link before sending the data, you can execute the Notify() method from the OnPoke subroutine, informing the client application that a change occurred. For information on hot links, see OnAdvise().

Example

See CLASS DDETOPIC for an example of using OnPoke property.

See Also

Advise(), Execute(), Initiate(), OnNewValue, OnPeek, Peek(), Poke(), Server, Terminate(), TimeOut, Topic, Unadvise()

OnRightDbIcIck

Executes a subroutine when the user double-clicks with the right mouse button while the pointer is on a form or an object.

Property of class

BROWSE, CHECKBOX, COMBOBOX, EDITOR, ENTRYFIELD, FORM, IMAGE, LISTBOX, PUSHBUTTON, RADIOBUTTON, RECTANGLE, SCROLLBAR, SPINBOX, TABBOX, TEXT

Data type

Function pointer or codeblock

Description

Use OnRightDbIcIck to perform an action when the user double-clicks with the right mouse button. OnRightDbIcIck can also trap *Shift*, *Ctrl*, left mouse button, or middle mouse button presses if they occur at the same time the user double-clicks the button.

OnRightDbIcIck passes three parameters to its subroutine:

- *<Flags>* is a single-byte value that tells you which other keys and mouse buttons were pressed when the user double-clicked the button.
- *<Col>* is the horizontal position of the mouse when the user double-clicked the button.
- *<Row>* is the vertical position of the mouse when the user double-clicked the button.

***<Flags>* parameter** If you want, you can use the *Flags* parameter with the BITSET() function to detect different key and mouse button combinations. BITSET() evaluates a numeric value and returns true (.T.) or false, depending on whether a particular bit in the value is on (1) or off (0). Give BITSET() two parameters:

- The *<Flags>* value itself
- The bit in *<Flags>* to evaluate

The values you can pass to BITSET() through the second parameter are as follows:

Simultaneous keypress	Second parameter
Left mouse button	0
<i>Shift</i>	2
<i>Ctrl</i>	3
Middle mouse button	4

For example, to determine if the user pressed *Shift* while clicking the object, use BITSET(Flags, 2). If the user did, BITSET() returns true. If BITSET(Flags, 3) and BITSET(Flags, 0) both return true, *Ctrl* and the left mouse button were pressed during the event. You can use such information in IF...ELSE...ENDIF or DO CASE...ENDCASE statements to assign different actions to different key and mouse button combinations.

<Col> and <Row> parameters The <Col> and <Row> parameters indicate where the right double-click event occurred, letting you make decisions accordingly. For example, the OnRightDbClick property of a form might execute different actions, depending on where the right double-click event occurred in the form. The <Col> and <Row> coordinates are expressed in character units.

Like other event properties, the OnRightDbClick property accepts:

- Functions
- Procedures
- Codeblocks

For information on these subroutines, see Chapter 4 in the *Programmer's Guide*. For information on writing event-handling subroutines, see Chapter 14 in the *Programmer's Guide*.

Note You can write a subroutine with the Procedure Editor, a window in which you enter dBASE program code. To access the Procedure Editor, click on the Tool button next to the OnRightDbClick item in the Inspector.

Example

NEW operator syntax:

```
LOCAL F1
F1=NEW Trips()
F1.OPEN()
CLASS Trips OF FORM
    this.Text = "Trip Schedule"
    this.Top = 2
    this.Left = 2
    this.Width = 38
    this.Height = 18
    this.OnRightDbClick = AnimalKingdom
* AnimalKingdom is a FUNCTION that performs
* an action and returns .T. or some value.
ENDCLASS
```

DEFINE object syntax:

```
DEFINE FORM Trips;
    PROPERTY Text "Trip Schedule",;
    Top 2, Left 2, Width 38, Height 18,;
    OnRightDbClick AnimalKingdom
OPEN FORM Trips
```

See Also

BITSET(), OnLeftDbClick, OnLeftMouseDown, OnLeftMouseUp, OnMiddleDbClick, OnMiddleMouseDown, OnMiddleMouseUp, OnRightMouseDown, OnRightMouseUp

OnRightMouseDown

Executes a subroutine when the user presses the right mouse button while the pointer is on a form or an object.

Property of class

BROWSE, CHECKBOX, COMBOBOX, EDITOR, ENTRYFIELD, FORM, IMAGE, LISTBOX, PUSHBUTTON, RADIOBUTTON, RECTANGLE, SCROLLBAR, SPINBOX, TABBOX, TEXT

Data type

Function pointer or codeblock

Description

Use OnRightMouseDown to perform an action when the user presses the right mouse button. OnRightMouseDown can also trap *Shift*, *Ctrl*, left mouse button, or middle mouse button presses if they occur at the same time the user presses the button.

OnRightMouseDown passes three parameters to its subroutine:

- *<Flags>* is a single-byte value that tells you which other keys and mouse buttons were pressed when the user pressed the button.
- *<Col>* is the horizontal position of the mouse when the user pressed the button.
- *<Row>* is the vertical position of the mouse when the user pressed the button.

<Flags> parameter If you want, you can use the *Flags* parameter with the BITSET() function to detect different key and mouse button combinations. BITSET() evaluates a numeric value and returns true (.T.) or false, depending on whether a particular bit in the value is on (1) or off (0). You give BITSET() two parameters:

- The *<Flags>* value itself
- The bit in *<Flags>* to evaluate

The values you can pass to BITSET() through the second parameter are as follows:

Simultaneous keypress	Second parameter
Left mouse button	0
<i>Shift</i>	2
<i>Ctrl</i>	3
Middle mouse button	4

For example, to determine if the user pressed *Shift* while clicking the object, use BITSET(Flags, 2). If the user did, BITSET() returns true. If BITSET(Flags, 3) and BITSET(Flags, 0) both return true, *Ctrl* and the left mouse button were pressed during the event. You can use such information in IF...ELSE...ENDIF or DO CASE...ENDCASE statements to assign different actions to different key and mouse button combinations.

<Col> and <Row> parameters The <Col> and <Row> parameters indicate where the right click event occurred, letting you make decisions accordingly. For example, the OnRightMouseDown property of a form might execute different actions, depending on where the right click event occurred in the form. The <Col> and <Row> coordinates are expressed in character units.

Like other event properties, the OnRightMouseDown property accepts:

- Functions
- Procedures
- Codeblocks

For information on these subroutines, see Chapter 4 in the *Programmer's Guide*. For information on writing event-handling subroutines, see Chapter 14 in the *Programmer's Guide*.

Note You can write a subroutine with the Procedure Editor, a window in which you enter dBASE program code. To access the Procedure Editor, click on the Tool button next to the OnRightMouseDown item in the Inspector.

Example

NEW operator syntax:

```
LOCAL F1
F1=NEW Trips()
F1.OPEN()
CLASS Trips OF FORM
    this.Text = "Trip Schedule"
    this.Top = 2
    this.Left = 2
    this.Width = 38
    this.Height = 18
    this.OnRightMouseDown = AnimalKingdom
* AnimalKingdom is a FUNCTION that performs
* an action and returns .T. or some value.
ENDCLASS
```

DEFINE object syntax:

```
DEFINE FORM Trips ;
    PROPERTY Text "Trip Schedule",;
    Top 2, Left 2, Width 38, Height 18,;
    OnRightMouseDown AnimalKingdom
OPEN FORM Trips
```

See Also

BITSET(), OnLeftDbClick, OnLeftMouseDown, OnLeftMouseUp, OnMiddleDbClick, OnMiddleMouseDown, OnMiddleMouseUp, OnRightDbClick, OnRightMouseUp

OnRightMouseUp

Executes a subroutine when the user releases the right mouse button while the pointer is on a form or an object.

Property of class

BROWSE, CHECKBOX, COMBOBOX, EDITOR, ENTRYFIELD, FORM, IMAGE, LISTBOX, PUSHBUTTON, RADIOBUTTON, RECTANGLE, SCROLLBAR, SPINBOX, TABBOX, TEXT

Data type

Function pointer or codeblock

Description

Use OnRightMouseUp to perform an action when the user releases the right mouse button. OnRightMouseUp can also trap *Shift*, *Ctrl*, left mouse button, or middle mouse button presses if they occur at the same time the user releases the button.

OnRightMouseUp passes three parameters to its subroutine:

- *<Flags>* is a single-byte value that tells you which other keys and mouse buttons were pressed when the user released the button.
- *<Col>* is the horizontal position of the mouse when the user released the button.
- *<Row>* is the vertical position of the mouse when the user released the button.

<Flags> parameter If you want, you can use the *Flags* parameter with the BITSET() function to detect different key and mouse button combinations. BITSET() evaluates a numeric value and returns true (.T.) or false (.F.), depending on whether a particular bit in the value is on (1) or off (0). Give BITSET() two parameters:

- The *<Flags>* value itself.
- The bit in *<Flags>* to evaluate.

The values you can pass to BITSET() through the second parameter are as follows:

Simultaneous keypress	Second parameter
Left mouse button	0
<i>Shift</i>	2
<i>Ctrl</i>	3
Middle mouse button	4

For example, to determine if the user pressed *Shift* while clicking the object, use BITSET(Flags, 2). If the user did, BITSET() returns true. If BITSET(Flags, 3) and BITSET(Flags, 0) both return true, *Ctrl* and the left mouse button were pressed during the event. You can use such information in IF...ELSE...ENDIF or DO CASE...ENDCASE statements to assign different actions to different key and mouse button combinations.

<Col> and <Row> parameters The *<Col>* and *<Row>* parameters indicate where the upward-right click event occurred, letting you make decisions accordingly. For

example, the OnRightMouseUp property of a form might execute different actions, depending on where the upward-right click event occurred in the form. The <Col> and <Row> coordinates are expressed in character units.

Like other event properties, the OnRightMouseUp property accepts:

- Functions
- Procedures
- Codeblocks

For information on these subroutines, see Chapter 4 in the *Programmer's Guide*. For information on writing event-handling subroutines, see Chapter 14 in the *Programmer's Guide*.

Note You can write a subroutine with the Procedure Editor, a window in which you enter dBASE program code. To access the Procedure Editor, click on the Tool button next to the OnRightMouseUp item in the Inspector.

Example

NEW operator syntax:

```
LOCAL F1
F1=NEW Trips()
F1.OPEN()
CLASS Trips OF FORM
  this.Text = "Trip Schedule"
  this.Top = 2
  this.Left = 2
  this.Width = 38
  this.Height = 18
  this.OnRightMouseUp = AnimalKingdom
* AnimalKingdom is a FUNCTION that performs
* an action and returns .T. or some value.
ENDCLASS
```

DEFINE object syntax:

```
DEFINE FORM Trips ;
  PROPERTY Text "Trip Schedule",;
  Top 2, Left 2, Width 38, Height 18,;
  OnRightMouseUp AnimalKingdom
OPEN FORM Trips
```

See Also

BITSET(), OnLeftDbClick, OnLeftMouseDown, OnLeftMouseUp, OnMiddleDbClick, OnMiddleMouseDown, OnMiddleMouseUp, OnRightDbClick, OnRightMouseDown

OnSelChange

Executes a subroutine when the highlight is moved from one prompt to another in a list box.

Property of class

LISTBOX, TABBOX

Data type

Function pointer or codeblock

Description

Use OnSelChange to execute a response each time a new list box prompt is selected.

OnSelChange lets you detect the user's movements through a list box. For example, a list box might use an OnSelChange subroutine to log the user's activity; the subroutine could store the prompt number (the current CurSel value) in a table each time the user moves from prompt to prompt.

Example

NEW operator syntax:

```
USE Contact
SET PRINTER TO FILE Record.Txt
* NEW Form definition (Lst)
Lst1=NEW LISTBOX(this)
Lst1.Width=40
Lst1.Height=15
Lst1.DataSource="FIELD Contact->Contact"
Lst1.OnSelChange={;? CompCode+SPACE(3)+Contact}
```

DEFINE object syntax:

```
USE Contact
SET PRINTER TO FILE Record.Txt
DEFINE FORM Lst FROM 1,1 TO 15,40
DEFINE LISTBOX Lst1 OF Lst;
  PROPERTY Width 40, Height 15;;
  DataSource "FIELD Contact->Contact";
  OnSelChange {;? CompCode+SPACE(3)+Contact}
OPEN FORM Lst
```

See Also

Selected(), Count(), CurSel

OnSelection

Executes a subroutine when the user submits a form.

Property of class

FORM

Data type

Function pointer or codeblock

Description

Use OnSelection to specify a subroutine or a codeblock to execute when the user submits a form. When the user selects the form, the ID property of the last object to have input focus is passed to the subroutine.

A form is submitted when the user

- Presses *Enter* when the form has focus and no browse object or editor object has focus.
- Presses *Spacebar* when a pushbutton has focus.
- Clicks a pushbutton.

When the subroutine you specify with OnSelection or ON SELECTION FORM finishes executing, program control returns to the form. However, if the subroutine executes the CLOSE FORM command, the form is closed.

OnSelection sends one parameter, *controlid*, to its subroutine. This parameter identifies which pushbutton submitted the form. The value in *controlid* is the same as the value in the ID property of the pushbutton.

Note You can use the ON SELECTION FORM command as an alternative to the OnSelection property.

Example

NEW operator syntax:

```
LOCAL F1
F1=NEW Trips()
F1.OPEN()
CLASS Trips OF FORM
  this.Text = "Trip Schedule"
  this.Top = 2
  this.Left = 2
  this.Width = 38
  this.Height = 18
  this.OnSelection = AnimalKingdom
  * AnimalKingdom is a FUNCTION that performs
  * an action and returns .T. or some value.
ENDCLASS
```


DEFINE object syntax:

```
DEFINE FORM Trips ;
  PROPERTY Text "Trip Schedule",;
  Top 2, Left 2, Width 38, Height 18,;
  OnSelection AnimalKingdom
```

See Also

OnGotFocus, ON SELECTION FORM

OnSize

Executes a subroutine after the user resizes a form.

Property of class

FORM

Data type

Function pointer or codeblock

Description

Use OnSize to perform actions automatically when the user resizes a form.

Like other event properties, the OnSize property accepts

- Functions
- Procedures
- Codeblocks

For information on these subroutines, see Chapter 4 in the *Programmer's Guide*. For information on writing event-handling routines, see Chapter 14 in the *Programmer's Guide*.

OnSize passes three parameters to its subroutine:

- *<nType>* is a numeric value that indicates how the user resizes the form. nType has five possible values:
 - 1 The user resized the form with the mouse or restored the form from a maximized or minimized condition.
 - 2 The user minimized the form.
 - 3 The user maximized the form.
 - 4 The form is not an MDI form, and the user maximized another non-MDI form.
 - 5 The form is not an MDI form, and the user minimized another non-MDI form.

The UTILS.H file provided with dBASE contains #define variables representing these *<nType>* values.

- *<width>* is the new width of the object.
- *<height>* is the new height of the object.

The height and width parameters reveal the new size, letting you make decisions accordingly.

Notes You can write a subroutine with the Procedure Editor, a window in which you enter dBASE program code. To access the Procedure Editor, click on the Tool button next to the OnSize item in the Inspector.

When the user resizes a form that does not have input focus initially, the When and OnGotFocus subroutines are executed first.

Before the user can resize a form manually, the Sizeable property must be set to true (.T.).

Example

NEW operator syntax:

```
LOCAL F1
F1=NEW Trips()
F1.OPEN()
CLASS Trips OF FORM
  this.Text = "Trip Schedule"
  this.Top = 2
  this.Left = 2
  this.Width = 38
  this.Height = 18
  this.OnSize = Shuffle
* Shuffle is a FUNCTION that performs
* an action and returns .T. or some value.
ENDCLASS
```

DEFINE object syntax:

```
DEFINE FORM Trips FROM 2,2 TO 20,40;
  PROPERTY Text "Trip Schedule",;
  Top 2, Left 2, Width 38, Height 18,;
  OnSize Shuffle
OPEN FORM Trips
```

See Also

OnChange, OnClick, OnClose, OnGotFocus, OnHelp, OnLeftMouseDown, OnLostFocus, OnMove, OnOpen, Sizeable

OnUnadvise

Executes a subroutine when dBASE is requested to stop notifying the client application when a dBASE data item changes.

Property of class

DDETOPIC

Data type

Logical

Description

Use the OnUnAdvise method to respond to the termination of a hot link. (A hot link lets you use the Notify() method to notify the client when an item changes.)

OnUnAdvise receives the *<item>* parameter, the name of a dBASE data item that the client application monitored. This item can be any dBASE field, variable, or array element.

Item names are often held in tables or array objects when multiple hot links are established. For example, when a client application requests a hot link to a field, it passes the field name to the OnAdvise subroutine. Each time, the OnAdvise subroutine can place the field name in an array object.

An OnPoke subroutine can search this array object each time a field is changed. If the name of the changed field is found in the array object, the routine can execute Notify(). The OnUnadvise() property can remove the field name from the array, preventing further notifications for that field.

Example

See CLASS DDETOPIC for an example of using OnUnAdvise property.

See Also

Advise(), Execute(), Initiate(), OnNewValue, Peek(), Poke(), Server, Terminate(), TimeOut, Topic, Unadvise()

Open()

Opens a form as a modeless window.

Property of class

FORM

Description

Use the Open() method to open a form and display its objects.

The form you open with Open() is *modeless*, and has the following characteristics:

- 1 While the form is open, focus can be transferred to other forms.
- 2 Execution of the routine that opened the form continues after the form is opened and active.

Open forms as modeless windows when you want more than one form open at once.

Notes To open a form as a *modal window*, use the ReadModal() method or the READMODAL() function. For example, forms that serve as dialog boxes are modal, since they halt program execution until the user inputs a response.

Open() is identical to the OPEN FORM command.

Example

```
LOCAL f
f = NEW XFORM()
f.Open()

CLASS XFORM OF FORM
  this.HelpFile = ""
  this.Top =      2.23
  this.Height =   20.00
  this.Text = "See..it opens"
  this.Left =     35.00
  this.Width =    73.80
  this.HelpId = ""
ENDCLASS
```

See Also

CLOSE..., Close(), OPEN FORM, ReadModal(), READMODAL()

PageCount()

Returns the highest numbered page defined for a form.

Property of class

FORM

Description

Use PageCount() to determine how many pages a multi-page form contains. For example, if you have a "Next Page" button or menu choice, you can use PageCount() in conjunction with PageNo to determine if you are already on the last page of a form.

If you have a form that is set up as a result of user input or other program activities, you might use PageCount() in conjunction with CLASS TABBOX to define a series of tabs for the defined pages.

Example

```
f = NEW Form()
DEFINE PUSHBUTTON p OF f;
PROPERTY;
    Height 2;;
    Left 2;;
    Top 2;;
    Text "Push",;
    Width 10
f.Open()
? f.PageCount()
```

See Also

CLASS TABBOX

PageNo

PageNo Returns the active page of a form or the page on which a control appears.

Property of class

BROWSE, CHECKBOX, COMBOBOX, EDITOR, ENTRYFIELD, FORM, IMGE, LINE, LISTBOX, OLE, PAINTBOX, PUSHBUTTON, RADIOBUTTON, RECTANGLE, SCROLLBAR, SHAPE, SPINBOX, TABBOX, TEXT

Data Type

Numeric

Default

There is no default for the PageNo property of a form. However, when you create a new form using the Form Designer, PageNo is set to 1.

The default for PageNo is 0 for a TabBox object and 1 for all other objects.

P

Description

The PageNo property of a form returns which page of the form is currently active. If you set a form's PageNo property to 0 (zero), all controls on all pages are displayed.

For all controls other than forms, the PageNo property specifies on which page of a multi-page form the control appears. A value of 0 (zero) means the control will appear on every page of the form.

You may want to implement multi-page forms whenever a single form contains a large number of objects. Dividing the objects logically among two or more pages helps organize the objects, and may make the form easier to use.

If you want to use tabs to let users switch pages, set the PageNo property of the TabBox to 0 (the default). This ensures that the tabs are visible while the user is on any page of the form. If you want any other control to appear on all pages (such as a Close button), set the PageNo property of the control to 0.

Example

The following example shows a form that contains two pages. The user switches between them by using buttons labeled Next Page and Previous Page. A Close button appears on every page.

```

LOCAL f
f = new MULTIPGFORM()
f.Open()
CLASS MULTIPGFORM OF FORM
  this.Top = 0
  this.Width = 60
  this.OnOpen = {;form.pageno=1}&& Make sure page 1 displays first
  DEFINE PUSHBUTTON PUSHBUTTON1 OF THIS;
    PROPERTY;
    Top 15;;
    PageNo 1;;
    Width 15;;
    Text "Next Page",;
    OnClick {;form.PageNo=2},;
    Left 42
  DEFINE PUSHBUTTON PUSHBUTTON2 OF THIS;
    PROPERTY;
    Top 15;;
    PageNo 2;;
    Width 15;;
    Text "Previous Page",;
    OnClick {;form.PageNo=1},;
    Left 42
  DEFINE PUSHBUTTON PUSHBUTTON3 OF THIS;
    PROPERTY;
    Top 17;;
    PageNo 0;;
    Width 10;;
    Text "Close",;
    OnClick {;form.Close()},;
    Left 42
  DEFINE TEXT TEXT1 OF THIS;
    PROPERTY;
    Top 4;;
    PageNo 1;;
    Width 34;;
    Text "This appears on page 1",;
    Height 2, ;
    Left 13
  DEFINE TEXT TEXT2 OF THIS;
    PROPERTY;
    Top 4;;
    PageNo 2;;
    Width 34;;
    Text "This appears on page 2",;
    Height 2, ;
    Left 13
ENDCLASS

```

See Also

CLASS TABBOX, PageCount(), SpeedBar

Parent

An object reference that points to the parent form of an object or a menu.

Property of class

BROWSE, CHECKBOX, COMBOBOX, EDITOR, ENTRYFIELD, IMAGE, LINE, LISTBOX, MENU, OLE, PUSHBUTTON, RADIOBUTTON, RECTANGLE, SCROLLBAR, SPINBOX, TABBOX, TEXT

Data type

Object reference

Description

Use Parent to reference the parent form of an object.

Set the Parent property with the OF *<form name>* clause of the DEFINE or REDEFINE command when you create the object. You can also set the Parent property with the *<form name>* parameter you specify for the object when you create the object with the NEW operator.

You can move a menu object from one form to another by changing the Parent property of the menu object. The Parent property is read-only for all other classes.

Example

The following example creates a form with a main menu presenting the text string "File". A submenu selection of File is "Change Parent Text", which uses the Parent property to change "File" to the string "New Text":

```
DEFINE FORM f1
  DEFINE MENU Main OF f1
  DEFINE MENU mFile OF f1.Main;
  PROPERTY;
  Text "File"
  DEFINE MENU mCp OF f1.Main.mFile;
  PROPERTY;
  Text "Change Parent Text",;
  OnClick {;this.Parent.Text = "New Text"}
OPEN FORM f1
```

P

See Also

ActiveControl, Before, First

Paste()

Copies text from the Windows clipboard to the currently active edit control.

Property of class

BROWSE, COMBOBOX, EDITOR, ENTRYFIELD, SPINBOX

Description

Use Paste() when the user wants to copy text from the Windows clipboard to the cursor position in the currently active edit control. The action of Paste() is identical to the Paste menu item on the standard Windows Edit menu.

If you have assigned a menubar to the form, you can use the menubar's EditPasteMenu property instead of using the Paste() property of individual objects on the form. For more information, see EditPasteMenu.

Example

See Copy() for an example.

See Also

Copy(), Cut(), EditPasteMenuUndo()

PatternStyle

Specifies a predefined Windows background hatching pattern.

Property of class

RECTANGLE

Data type

Numeric

Default

The default for PatternStyle is 0.

Description

Use PatternStyle to select a predefined Windows background hatching pattern for a rectangle object.

You can specify the following settings for PatternStyle








Table 8.1 Fill patterns		
Number	Description	Example
0	Solid	
1	BDiagonal	

Table 8.1 Fill patterns

Number	Description	Example
2	Cross	
3	Diagcross	
4	FDiagonal	
5	Horizontal	
6	Vertical	

Example

NEW operator syntax:

```
Comp = NEW ENTRYFIELD(this)
Comp.Datalink = "Clients->Company"
Comp.PatternStyle = 3
```

DEFINE object syntax:

```
DEFINE ENTRYFIELD Comp OF THIS;
PROPERTY Datalink "Clients->Company", ;
PatternStyle 3
```

See Also

ColorHighlight, ColorNormal

Peek()

Retrieves a data item from a DDE server.

Property of class

DDELINK

Description

Use the Peek() method to read data from a DDE server topic.

A server topic is usually a file you open in an external application. For example, a data-exchange program might start a session in Quattro Pro for Windows, open one of its spreadsheet files (the topic), and use Peek() to read one of its cells.

Peek() requires *<item>*, a parameter that identifies a data item in the server topic. This item can be any single element, such as a field in a table or a cell in a spreadsheet. For example, you can read cell C2 of Page A in a Quattro Pro spreadsheet file by passing the *<item>* parameter "A:C2".

Before you can query data from a server topic, you need to establish a DDE link to it. For information on establishing DDE links, see Initiate() and Chapter 26 in the *Programmer's Guide*.

P

Example

```
PUBLIC LinkObj
LinkObj = NEW DDELINK()
IF LinkObj.Initiate("QPW","Demo.WB1")
    ? "Connection to QPW initiated"
ELSE
    ? "Connection to QPW failed"
ENDIF
mValue1=LinkObj.Peek("A:A1")
? mValue1
```

See Also

Advise(), Execute(), Initiate(), OnNewValue, Poke(), Server, Terminate(), TimeOut, Topic, Unadvise()

Pen

Specifies the pattern of a line object.

Property of class

LINE

Data type

Numeric

Default

The default for Pen is 0 (Solid).

Description

Use Pen to control the appearance of a line object.

You can specify any of five settings for Pen:

Table 8.2 Pen patterns		
Number	Description	Example
0	Solid	_____
1	Dash	- - - - -
2	Dot
3	Dash Dot	- . - . - .
4	DashDotDot	- . - . - .

Example

NEW operator syntax:

```
Ln2=NEW LINE(this)
Ln2.Left=3
Ln2.Top=8
Ln2.Bottom=8
Ln2.Right=33
Ln2.Pen = 3                && Dash Dot
```

DEFINE object syntax:

```
DEFINE LINE Ln2 OF THIS;
  PROPERTY Left 3, Top 8, Bottom 8,;
           Right 33, Pen 3      && Dash Dot
```

See Also

PatternStyle

PenStyle

Specifies the type of line to be used as the border of a shape object.

Property of class

SHAPE

Data type

Numeric

Default

The default for PenStyle is 0 (Solid).

Description

Use `PenStyle` to control the appearance of the border of a shape object.

You can specify any of five settings for PenStyle:

Number	Description	Example
0	Solid	_____
1	Dash	- - - - -
2	Dot
3	Dash Dot	- . - . - .
4	DashDotDot	- . . - . .

Example

NEW operator syntax:

```
Sh2=NEW SHAPE(this)
Sh2.Left=3
Sh2.Top=8
Sh2.PenStyle = 3                && Dash Dot
DEFINE object syntax:
DEFINE SHAPE Sh2 OF THIS;
    PROPERTY Left 3, Top 8, PenStyle 3  &&Dash Dot
```

See Also

PenWidth, ShapeStyle

PenWidth

Specifies the width in pixels of the line used as the border of a shape object.

Property of class

SHAPE

Data type

Numeric

Default

The default for PenWidth is 1.

Description

Use PenWidth to specify the thickness of the line used to border a shape object. If you set PenWidth to a value greater than 1, then PenStyle can only be set to 0.

Example

NEW operator syntax:

```
Sh2=NEW SHAPE(this)
Sh2.Left=3
Sh2.Top=8
Sh2.PenWidth = 3                && 3 pixels
DEFINE object syntax:
DEFINE SHAPE Sh2 OF THIS;
    PROPERTY Left 3, Top 8, PenWidth 3  && 3 pixels
```

See Also

PenStyle, ShapeStyle

Picture

Formats text in an entry field, a spin box, or a text object.

Property of class

ENTRYFIELD, SPINBOX, TEXT

Data type

Character

Default

The default for Picture is an empty string.

Description

Specify the picture setting with a character string called a *template*. A template can consist of

- 1 Picture template characters, which represent and modify individual characters in the text string.
- 2 Function symbols, which usually modify the entire text string. (For information on function symbols, see the Function property.)
- 3 Literal characters, which are inserted into the text string.

Here are the picture template characters:

9	Restricts entry of character data to numbers, and restricts entry of numeric data to numbers and + and - signs
#	Restricts entry to numbers, spaces, periods, and signs
!	Converts letters to uppercase
\$	Inserts a dollar sign or the symbol defined with SET CURRENCY TO instead of leading blanks
*	Inserts asterisks in place of leading spaces
.	Marks the position of the decimal point
,	Separates thousands with a comma (or with another character indicated by SET SEPARATOR)
A	Restricts entry to alphabetic characters
L	Restricts entry to T, t, F, f, Y, y, N, or n, and converts it to uppercase
N	Restricts entry to letters and numbers
X	Allows any character
Y	Restricts entry to Y, y, N, or n, and restricts display to Y and N

If you use function symbols in a template, precede them with the @ symbol. If you combine template characters and function symbols in the same template, list function symbols first and separate them from the template characters with a space.

Note You can specify a picture template with the Choose Template dialog box, which lets you enter picture templates. To access the Choose Template dialog box, click the Tool button next to the Picture item in the Inspector.

Example

NEW operator syntax:

```

FLD1.Picture = "@"!
* or
FLD1.Picture = "$999,999,999.99"
* or
FLD1.Picture = "99:99:99"

```

DEFINE object syntax:

```

DEFINE ENTRYFIELD FLD1 OF THIS;
  PROPERTY Datalink "<char, numeric or date field>;"
  Picture "@"!
* or
  Picture "$999,999,999.99"
* or
  Picture "99:99:99"

```

See Also

@...SAY...GET, DEFINE, Function, TRANSFORM()

Poke()

Inserts data into a server document.

Property of class

DDELINK

Description

Use the Poke() method to write data to a server document.

A server document is a file you open in an external application. For example, a data-exchange program might start a session in Quattro Pro for Windows, open one of its spreadsheet files, and use Poke() to write a value into one of its cells.

Poke() requires two parameters:

- *<item>* is the element you write to in the server document. It can be any single item, such as a field in a table or a cell in a spreadsheet. For example, "B:G2" specifies Cell G2 of Page B in a Quattro Pro spreadsheet file.
- *<value>* is the value you write to the server document. For example, you can pass a literal or the value stored in a field.

Before you can send data to a server document, you need to open the server application, open the document, and establish a DDE link to it. For information on establishing DDE links, see Initiate() and Chapter 26 in the *Programmer's Guide*.

Example

```
PUBLIC LinkObj
LinkObj = NEW DDELINK()
IF LinkObj.Initiate("QPW","Demo.WB1")
    ? "Connection to QPW initiated"
ELSE
    ? "Connection to QPW failed"
ENDIF
LinkObj.Poke("A:A3","198")
mValue2=LinkObj.Peek("A:A3")
? mValue2           && Confirm success of .POKE()
```

See Also

Advise(), Execute(), Initiate(), OnNewValue, Peek(), Server, Terminate(), TimeOut, Topic, Unadvise()

PopupMenu

Specifies a popup menu for a form.

Property of class

FORM

Data Type

Object reference

Description

Assign the PopupMenu property to an existing popup menu to have the popup appear when the user right clicks on the form.

Example

The following example creates a form and attaches an existing popup menu (Pop0328.pop) to the form's OnOpen property. It then opens the form and the Inspector, so you can inspect the form's PopupMenu property. If you right-click while the form is open, the popup menu is displayed.

```
f = new POPFORM()
f.Open()
INSPECT(f)
CLASS POPFORM OF FORM
    this.TopMost = .F.
    this.Height = 20
    this.Left = 53
    this.Top = 0
    this.PageNo = 1
    this.Width = 60
    this.OnOpen = CLASS::FORM_ONOPEN
    Procedure Form_OnOpen
```

P

Print()

```
IF TYPE("This.PopupMenu") # "O"  
DO Pop0328.pop with this,"MyPopTest"  
form.PopupMenu = form.MyPopTest  
ENDIF  
Return  
ENDCLASS
```

The code in Pop0328.pop was generated by the Menu Designer:

```
* Pop0328.pop  
Parameter FormObj,PopupName  
NEW POP0328MENU(FormObj,PopupName)  
CLASS POP0328MENU(FormObj,PopupName) OF POPUP(FormObj,PopupName)  
this.Top = 0  
this.Left = 0  
this.TrackRight = .T.  
DEFINE MENU CLOSE OF THIS;  
PROPERTY;  
Text "Close",;  
OnClick {;form.close()  
ENDCLASS
```

See Also

OnOpen, TrackRight

Print()

Prints a form and the objects it contains.

Property of class

FORM

Description

Use the Print() method to print a form on a selected printer.

Executing the Print() method opens the Print dialog box, which lets the user determine:

- The number of pages to print
- The print quality
- The number of copies to print
- If the output goes to the printer or to a file
- If the output is collated

The user clicks the OK button to print the form.

Executing the Print() method is equivalent to selecting File | Print. To determine which printer receives the output, execute the CHOOSEPRINTER() function.

Example

The following example defines a form that includes a browse object. When the user has located a record or subset of records to print, clicking the right mouse calls a codeblock that uses PRINT() to send currently displayed records to the printer:

```
LOCAL f
f=NEW EntryForm()
f.OPEN()
CLASS EntryForm OF FORM
  this.Top=2
  this.Left=2
  this.Width=72
  this.Height=20
  this.View = "Animals.DBF"
  this.OnRightMouseDown = {;this.Print()}
DEFINE BROWSE Br1 OF THIS;
  PROPERTY Top 2, Left 1,;
  Width 50, Height 10
ENDCLASS
```

See Also

CHOOSEPRINTER(), PRINTJOB...ENDPRINTJOB, SET PRINTER

RangeMax

Determines the upper limit for values the user can enter in a spin box.

Property of class

SCROLLBAR, SPINBOX

Data type

Date or numeric

Default

The default for RangeMax is 100.00.

Description

Use RangeMax in combination with RangeMin to specify a range restriction for values entered into a scroll bar or a spin box. (RangeMax sets the upper limit and RangeMin sets the lower limit.) For example, an application that lets the user input a percentage might prevent the input of a value less than 0 or greater than 100.

Users must enter a value within the specified range before they can give focus to another object.

Note Range restrictions have effect only when the RangeRequired property is true (.T.).

R

Example

NEW operator syntax:

```
SPNBX1.RangeMax = 100
* or
SPNBX1.RangeMax = {12/31/94}
```

DEFINE object syntax:

```
DEFINE SpinBox Spn1 OF THIS;
  PROPERTY DataLink "<numeric or date field>";
  RangeMax 100
* or
  RangeMax {12/31/94}
```

See Also

RangeMin, RangeRequired, Valid, ValidErrorMsg, ValidRequired

RangeMin

Determines the lower limit for values the user can enter in a spin box.

Property of class

SCROLLBAR, SPINBOX

Data type

Date or numeric

Default

The default for RangeMin is 1.

Description

Use RangeMin in combination with RangeMax to specify a range restriction for values entered into a scroll bar or a spin box. (RangeMin sets the lower limit and RangeMax sets the upper limit.) For example, an application that lets the user input a percentage might prevent the input of a value less than 0 or greater than 100.

Users must enter a value within the specified range before they can give focus to another object.

Note Range restrictions have effect only when the RangeRequired property is true (.T.).

Example

NEW operator syntax:

```
SPNBX1.RangeMin = 1
* or
SPNBX1.RangeMin = {|01/01/94}
```

DEFINE object syntax:

```

DEFINE SpinBox Spn1 OF THIS;
  PROPERTY Datalink "<numeric or date field>;";
  RangeMin 1
  * or
  RangeMin {|01/01/94}

```

See Also

RangeMax, RangeRequired, Valid, ValidErrorMsg, ValidRequired

RangeRequired

Determines whether the range you specify with the RangeMax and RangeMin properties is enforced every time the control receives focus, or only when the user makes changes.

Property of class

SPINBOX

Data type

Logical

Default

The default for RangeRequired is false (.F.).

Description

Set RangeRequired to true (.T.) when you want to enforce a range limitation specified by the RangeMax and RangeMin properties on previously entered data as well as on new data. When RangeRequired is false (the default), RangeMin and RangeMax settings are not checked if the user doesn't change existing values.

For example, when the RangeRequired property of a spin box is set to true, dBASE detects any range error and forces the user to correct the problem. Users must enter a value within the specified range before they can give focus to another object.

Example

NEW operator syntax:

```

Date = NEW ENTRYFIELD(this)
Date.Datalink = "Clients->Baldate"
Date.RangeRequired = .T.
Date.RangeMax = {|12/31/95}

```

DEFINE object syntax:

```

DEFINE ENTRYFIELD Date OF THIS;
  PROPERTY Datalink "Clients->Baldate";
  RangeRequired .T., RangeMax {|12/31/95}

```

R

See Also

RangeMax, RangeMin, Valid, ValidErrorMsg

ReadModal()

Opens a form as a modal window.

Property of class

FORM

Description

Use ReadModal() to open a form as a modal window. A modal window has the following characteristics:

- 1 While the form is open, focus can't be transferred to other forms.
- 2 Execution of the routine that opened the form stops until the form is closed. When the form is closed, control transfers to the command line after the one that opened the form.

Many applications use modal forms as dialog boxes, which typically require users to take an action before the dialog box can be closed.

You can't open a form with the ReadModal() method or the READMODAL() function when the MDI property is set to true.

To open a form as a *modeless* window, use the Open() method or the OPEN FORM command.

The ReadModal() method is identical to the READMODAL() function.

Example

```
LOCAL f
f=NEW EntryForm()
f.readmodal()
CLASS EntryForm OF FORM
  this.MDI=.F.
  this.Top=2
  this.Left=2
  this.Width=38
  this.Height=13
  * subsequent object definitions
ENDCLASS
```

See Also

Close(), CLOSE..., OPEN FORM, Open(), READMODAL()

Reconnect()

Attempts to restart a terminated conversation with a DDE server application and returns true (.T.) if successful.

Property of class

DDELINK

Description

Use Reconnect() to restore a DDE link that was terminated with the Terminate() method.

Use a DDE link to exchange data and instructions with another application. For example, a data-exchange program might establish a link with Quattro Pro for Windows and open one of its spreadsheet files, then copy data from the spreadsheet to a dBASE table.

When you terminate a DDE link with Terminate(), you can restore it with Reconnect. When you terminate the link with the Release() method, the link can't be restored and you need to create the DDELink object again.

Example

```
PUBLIC LinkObj
LinkObj = NEW DDELINK()
LinkObj.Initiate("QPW", "Demo.WB1")
* Subsequent program operations completed; link
* do longer required
LinkObj.Terminate()
* Later in program, DDE link object again needed
LinkObj.Reconnect()
```

See Also

Initiate(), Release(), Terminate()

Refresh()

R

Updates data displayed in control objects within a form.

Property of class

FORM

Description

Use Refresh() to update data displayed in a form to reflect the current state of the data as it exists on disk. For example, you can use Refresh() in a multi-user environment to ensure that the displayed data reflects all recent changes made by other users.

Refresh()

Example

The following example lets you use a scrollbar or an entry field to change the data in the StartBal field of the Clients table. Because Refresh() is assigned to the scrollbar's OnChange property, the value in the entry field and the table always reflect the value as chosen with the scrollbar.

```
LOCAL f
f = NEW SBAR2FORM()
f.Open()
CLASS SBAR2FORM OF FORM
  this.Top = 0
  this.PageNo = 1
  this.Width = 49
  this.View = "CLIENTS.DBF"
  this.Height = 20
  this.Left = 50
  DEFINE BROWSE BROWSE1 OF THIS;
    PROPERTY;
      Top 2;;
      PageNo 1;;
      Width 25.835;;
      CUATab .T.;;
      Alias "CLIENTS",;
      ScrollBar 2;;
      Fields "CLIENT_ID,STARTBAL",;
      Height 11;;
      Left 6;;
      ShowRecNo .F.
  DEFINE SCROLLBAR SCROLLBAR1 OF THIS;
    PROPERTY;
      Top 16;;
      PageNo 1;;
      Width 20;;
      ColorNormal "ScrollBar",;
      OnChange {form.refresh()},;
      DataLink "CLIENTS->STARTBAL",;
      Height 1;;
      Vertical .F.;;
      Left 9, ;
      Rangemax 32766

  DEFINE ENTRYFIELD ENTRYFIELD1 OF THIS;
    PROPERTY;
      Top 15;;
      PageNo 1;;
      Width 11;;
      ColorNormal "WINDOWTEXT/WINDOW",;
      Border .T.;;
      DataLink "CLIENTS->STARTBAL",;
      Height 1;;
      Left 14.5
  Procedure SCROLLBAR1_OnChange

  form.refresh()
ENDCLASS
```

See Also
REFRESH, SHOW OBJECT

Release()

Removes an object definition from memory.

Property of class

BROWSE, DDELINK, DDETOPIC, CHECKBOX, COMBOBOX, EDITOR, ENTRYFIELD, FORM, IMAGE, LINE, LISTBOX, MENU, OLE, PUSHBUTTON, RADIOBUTTON, RECTANGLE, SCROLLBAR, SPINBOX, TABBOX, TEXT

Description

Use Release() to conserve memory resources when an object is no longer needed. For example, when an application doesn't need a form, removing the form from memory frees up memory for other purposes.

Note When a form is closed, dBASE releases it from memory automatically if there is no object reference pointing to it.

Example

NEW operator syntax:

```
LOCAL f
f=NEW EntryForm()
f.OPEN()
CLASS EntryForm OF FORM
  this.Top=2
  this.Left=2
  this.Width=38
  this.Height=13
  this.OnClose={;Form.Release()}
* Subsequent object definitions
ENDCLASS
```

DEFINE object syntax:

```
DEFINE FORM EntryForm;
  PROPERTY Top 2, Left 2, Width 38, Height 13,;
  OnClose {;Form.Release()}
* Other control definitions
OPEN FORM EntryForm
```

R

See Also
RELEASE OBJECT

RemoveAll()

Deletes all elements from an associated array object.

Property of class

ASSOCARRAY

Description

Use the RemoveAll() method to remove all elements from an associated array. You might want to do this if you want to repopulate the array with new values.

Example

The following example removes all elements from an associated array.

```
aa = NEW ASSOCARRAY()
aa["USA"] = "Washington, DC"
aa["Spain"] = "Madrid"      && Array contains two elements
aa.RemoveAll()              && Array now contains no elements
```

See Also

IsIndex(), RemoveKey()

RemoveKey()

Deletes an element from an associated array object.

Property of class

ASSOCARRAY

Description

Use the RemoveKey() method to remove elements from an associated array object. RemoveKey() accepts a character string as its parameter. This string represents the subscript of the associated array element you want to remove.

Example

The following example removes an element from an associated array.

```
aa = NEW ASSOCARRAY()
aa["USA"] = "Washington, DC"
aa["Spain"] = "Madrid"      && Array contains two elements
aa.RemoveKey("USA")         && Array now contains one element
```

See Also

Delete(), IsIndex(), RemoveAll()

Resize()

Increases or decreases the number of elements in an array object.

Property of class

ARRAY

Description

Use the Resize() method to add or remove rows and columns from an array object. The Resize() method is similar to the ARESIZE() function.

Resize() accepts three parameters:

- *<new rows expN>*—The number of rows in the resized array object. *<new rows expN>* must always be a positive, nonzero value.
- *<new cols expN>*—The number of columns in the resized array object. *<new cols expN>* must always be 0 or a positive value. If you omit this option, Resize() changes the number of rows and leaves the number of columns the same.
- *<retain values expN>*—Determines how the array elements are rearranged when rows are added or removed. To see the effect of *<retain values expN>*, see ARESIZE().

Example

USE Customer.DBF

```
* Initialize array object
ObjArr=NEW ARRAY(RECCOUNT())
* Fill 1-dimensional array with values
* from Name field of Customer.DBF
GO TOP
FOR i=1 TO RECCOUNT()
    ObjArr[i]=Customer->Name
    SKIP
Next i
* Use RESIZE() to add two additional columns to the
* existing array and enter Ytd_Sales and phone data
* in the new columns.
ObjArr.RESIZE(RECCOUNT(),3,1)
GO TOP
FOR i=1 TO RECCOUNT()
    ObjArr[i,2]=Customer->YTD_Sales
    ObjArr[i,3]=Customer->Phone
    SKIP
Next i
* Display Contents of 3-dimensional array
FOR i = 1 TO RECCOUNT()
    ? ObjArr[i,1], ObjArr[i,2], ObjArr[i,3]
Next i
```

R

See Also

Add(), Insert(), GROW(), ARESIZE(), INSERT()

Right

Specifies the position of the right end of a line object relative to its parent form.

Property of class

LINE

Data type

Numeric

Description

Use the Right property in combination with the Bottom, Left, and Top properties to determine the position and length of a line object.

Each unit of the value you assign to Right is the average width of characters in the active font of the parent form. For example, if you set the Right property of a line to 20, the right end of the line is positioned 20 characters to the right.

Example

```

DEFINE LINE Ln1 OF THIS;
PROPERTY;
  Left 10;;          && Vertical line from 3,10
  Top 3;;            && to 8,10
  Width 4;;
  Bottom 8;;
  ColorNormal "RB"
DEFINE LINE Ln2 OF THIS;
PROPERTY;
  Left 3;;           && Horizontal line from 8,3
  Top 8;;            && to 8,33
  Width 4;;
  Bottom 8;;
  Right 33;;
  ColorNormal "RB"

```

See Also

Height, ScaleFontName, ScaleFontSize, Top, Width

SaveRecord()

Saves a temporary record by appending it to the currently active table.

Property of class

FORM

Description

Use SaveRecord() to add to the currently active table a new record stored in a temporary memory buffer you created with BeginAppend().

For more information, see BeginAppend().

Example

See BeginAppend() for an example.

See Also

AbandonRecord(), BeginAppend(), IsRecordChanged()

ScaleFontName

Determines which font the coordinate plane of the form is based on.

Property of class

FORM

Data type

Character

Default

The default for ScaleFontName is MS Sans Serif.

Description

Use ScaleFontName in combination with ScaleFontSize to determine the height of rows and the width of columns in the *coordinate plane* of a form.

The coordinate plane is a two-dimensional grid of row and column coordinates. The height of rows and the width of columns in the coordinate plane depend primarily on

- The active font of the form, which you specify with ScaleFontName. Different fonts have different widths and heights, so changing the specification in ScaleFontName can alter the height of rows and the width of columns.
- The size of the font, which you specify with ScaleFontSize. (The value you specify for ScaleFontSize is in points).

S

Together, the row height and column width of a coordinate plane make up a *character unit*. Properties such as Height and Width use character units to determine object size and position. For example, the Height property expresses height by character units, as with:

```
MyForm.MyObj.Height = 2.4    && 2.4 character units
```

The actual height of the object depends on the number of character units *and* on character unit size.

For more information on the coordinate plane, see Chapter 16 in the *Programmer's Guide*.

Note You can select a font with the Font dialog box, which displays all installed fonts. To access the Font dialog box, click on the Tool button next to the ScaleFontName item in the Inspector. You can also open the Font dialog box with the GETFONT() function.

Example

```
LOCAL f
f=NEW Scale()
f.OPEN()
CLASS Scale OF FORM
  this.ScaleFontName = "Courier"
  this.ScaleFontSize = 15
  DEFINE TEXT Txt1 OF THIS;
    PROPERTY Text "Visual dBASE",;
    Width 40, Top 5, Alignment 4
ENDCLASS
```

See Also

Bottom, GetTextExtent(), Height, Right, Top, Width

ScaleFontSize

Determines the height of each row and the width of each column in the coordinate plane of a form.

Property of class

FORM

Data type

Numeric

Default

The default for ScaleFontSize is 8.00.

Description

Use ScaleFontSize in combination with ScaleFontName to determine the height of rows and the width of columns in the *coordinate plane* of a form.

The coordinate plane is a two-dimensional grid of row and column coordinates. The height of rows and the width of columns in the coordinate plane depend primarily on

- The active font of the form, which you specify with `ScaleFontName`. Different fonts have different average widths and heights, so changing the specification in `ScaleFontName` can alter the height of rows and the width of columns.
- The size of the font, which you specify with `ScaleFontSize`. (The value you specify for `ScaleFontSize` is in points).

Together, the row height and column width of a coordinate plane make up a *character unit*. Properties such as `Height` and `Width` use character units to determine object size and position. For example, the `Height` property expresses height by character units, as with:

```
MyForm.MyObj.Height = 4.5    && 4.5 Character units
```

The actual height of the object depends on the number of character units *and* on character unit size.

For more information on the coordinate plane, see Chapter 16 in the *Programmer's Guide*.

Example

```
LOCAL f
f=NEW Scale()
f.OPEN()
CLASS Scale OF FORM
  this.ScaleFontName = "Courier"
  this.ScaleFontSize = 15
  DEFINE TEXT Txt1 OF THIS;
    PROPERTY Text "dBASE for Windows",;
    Width 40, Top 5, Alignment 4
ENDCLASS
```

See Also

Bottom, Height, Right, Top, Width

Scan()

Searches an array object for a specified value.

S

Property of class

ARRAY

Description

Use the `Scan()` method to search an array object for a value. For example, if an array object contains customer names, use `Scan()` to find the location in which a particular name appears. The `Scan()` method is similar to the `ASCAN()` function.

Scan() returns the element number of the first element that matches the expression if the search is successful, or 0 if the search is unsuccessful. If necessary, use Subscript() to determine the subscript numbers of the element. For more information about element numbers and subscripts, see Element() and Subscript().

Scan() accepts three parameters:

- *<exp>* is the expression to search for.
- *<starting element expN>* is the element number of the element at which to start searching. Without *<starting element expN>*, Scan() starts searching at the first element.
- *<elements expN>* is the number of elements that Scan() searches. Without *<elements expN>*, Scan() searches the array object from *<starting element expN>* to the last element. If you specify a value for *<elements expN>*, also specify a value for *<starting element expN>*.

When *<exp>* contains string data, Scan() is case-sensitive, so you might want to use UPPER(), LOWER(), or PROPER() to match the case of *<exp>* with the case of the data stored in the array object.

When *<exp>* contains string data, Scan() searches for an expression following the rules established by SET EXACT. If SET EXACT is ON, dBASE returns 0 if the value in *<exp>* is not identical to the data in an element of the array object. If SET EXACT is OFF, dBASE returns 0 if the characters in *<expN>* do not match the beginning characters in the data in an element of the array object. For more information, see SET EXACT.

Example

```
USE Animals.DBF
* Initialize an array object
ObjArr=NEW ARRAY(RECCOUNT(),3)
* Fill array with values from Animals.DBF
COPY TO ARRAY ObjArr FIELDS Name, Area, Weight
* Use SCAN() to search the array for the string
* "Parrot" and return the array element number.
* SUBSCRIPT() is used to return the row and column
* of the element number returned by SCAN():
String = "Parrot"
aElement = ObjArr.Scan(String)
aRow = ObjArr.SUBSCRIPT(aElement,1)
aCol = ObjArr.SUBSCRIPT(aElement,2)
? String + " is located at Row " + ;
  LTRIM(STR(aRow)) + ", Column " + ;
  LTRIM(STR(aCol))
```

See Also

ASCAN(), Element(), LOWER(), PROPER(), SET EXACT, Sort(), Subscript(), UPPER()

ScrollBar

Determines if an object has a scroll bar.

Property of class

BROWSE, EDITOR, FORM

Data type

Numeric

Default

The default for Scrollbar is 1 (On) for editor objects, 0 (Off) for forms, and 2 (Auto) for browse objects.

Description

Create a scroll bar to let the user page through a form or through a browse or editor object, when its contents exceed its size. For example, if an editor object contains twenty lines of text and the editor object is only ten lines high, a scroll bar lets the user page up and down through the text.

Scrollbar can have any of four settings:

ScrollBar value	Description
0 (Off)	The object has no scroll bar.
1 (On)	The object has a scroll bar.
2 (Auto)	Displays the scroll bar only when needed.
3 (Disabled)	The scroll bar is visible but it's not usable.

Example

```
USE Contact.DBF
LOCAL f
f=NEW ENTRY()
f.OPEN()
CLASS Entry OF FORM
  this.Top=2
  this.Left=2
  this.Width=72
  this.Height=20
  this.ScrollBar=1
  DEFINE EDITOR ED1 OF THIS;
    PROPERTY Top 4,Left 37,Width 32,;
      Height 12,ScrollBar 0,;
      DataSource "MEMO Contact->Notes"
ENDCLASS
```

See Also

CLASS SCROLLBAR

SelectAll

Determines if the value contained in an entry field or a spin box initially appears selected (highlighted).

Property of class

ENTRYFIELD, SPINBOX

Data type

Logical

Default

The default for SelectAll is true (.T.).

Description

Set SelectAll to true (.T.) to give the user a shortcut for deleting or replacing the initial value in an entry field or a spin box. The value (which you specify with the DataLink or Value property) is highlighted when the user gives the object focus, and the first character the user enters overwrites the value. Pressing *Del* or *Backspace* deletes the value. Pressing a direction key (such as *Left arrow* or *Right arrow*) removes the highlight without erasing the value.

Example

NEW operator syntax:

```
Date = NEW ENTRYFIELD(this)
Date.Datalink = "Clients->Baldate"
Date.SelectAll = .T.
```

DEFINE object syntax:

```
DEFINE ENTRYFIELD Date OF THIS;
PROPERTY Datalink "Clients->Baldate",;
SelectAll .T.
```

See Also

Mode, Style

Selected()

Returns the currently selected prompt in a list box.

Property of class

LISTBOX

Description

Use Selected() in combination with Count() to evaluate user choices in a multiple-choice list box. For example, you can see which prompts were chosen by evaluating each prompt with the Selected() method in a FOR...NEXT loop. (If the list box has an uncertain number of prompts, use Count() to determine the number of times to execute the loop.)

Selected() accepts one optional parameter, *<item>*, a numeric value that identifies an item by its relative position in the list box.

Make a list box multiple-choice by setting the Multiple property to true (.T.).

Example

See Count() for an example of using Selected().

See Also

FOR...NEXT, Count(), Multiple

Separator

Determines if a menu item is a line that the user can't select.

Property of class

MENU

Data type

Logical

Default

The default for Separator is false (.F.).

Description

Set Separator to true (.T.) when you want to use a menu item as a separator between groups of menu commands. For example, a menu titled Accounting might use a separator to emphasize the distinction between Accounts Receivable items and Accounts Payable items.

Example

```

DEFINE FORM f1
DEFINE MENU Main OF f1
DEFINE MENU mOpt1 OF f1.Main;
  PROPERTY;
  Text "Option 1"
DEFINE MENU mSlct1 OF f1.Main.mOpt1;
  PROPERTY;
  Text "Select 1"
DEFINE MENU mLine1 OF f1.Main.mOpt1;
  PROPERTY;
  Separator .T.
DEFINE MENU mSlct2 OF f1.Main.mOpt1;
  PROPERTY;
  Text "Select 2"
DEFINE MENU mLine2 OF f1.Main.mOpt1;
  PROPERTY;
  Separator .T.
DEFINE MENU mSlct3 OF f1.Main.mOpt1;
  PROPERTY;
  Text "Select 3"
OPEN FORM f1

```

See Also

CLASS MENU

Server

Holds the name of a DDE server application.

Property of class

DDELINK

Data type

Character

Description

Use Server to identify the server application that you established a DDE link to.

Create a DDE link with the Initiate() method. For example, Initiate() might establish a link to Quattro Pro for Windows, open one of its spreadsheet files, and copy data from its cells into a dBASE table.

Server holds the value of the first parameter you passed to Initiate(), which is the name of the main executable file of the server application.

Example

```

PUBLIC LinkObj
LinkObj = NEW DDELINK()
LinkObj.Initiate("QPW", "Demo.WB1")
* Subsequent program operations
IF LinkObj.Server="QPW" .AND. ;
    LinkObj.Topic = "Demo.WB1"
    mValue1=LinkObj.Peek("A:A1")
ENDIF

```

See Also

Advise(), Execute(), Initiate(), OnNewValue, Peek(), Poke(), Terminate(), TimeOut, Topic, Unadvise()

ServerName

Identifies the server application that is invoked when the user double-clicks an OLE viewer object.

Property of class

OLE

Data type

Character

Default

The default for ServerName is an empty string.

Description

Use ServerName to anticipate which server application is activated if the user double-clicks the current OLE viewer object.

An OLE viewer object displays an OLE document. An OLE document can be a graphic image, a document created by a word processor, or any other data object created by an external application. This external application is known as the *OLE server*. For example, a graphic image created in Paintbrush can be an OLE document, and Paintbrush can be an OLE server, if you embed the graphic image (or a portion of it) in an OLE field with the Cut and Paste commands of the Edit menu.

ServerName is a read-only property.

Example

See DoVerb() for an example of using ServerName.

See Also

LinkFileName, OleType

SetFocus()

Gives focus to a form or an object in a form.

Property of class

BROWSE, CHECKBOX, COMBOBOX, EDITOR, ENTRYFIELD, FORM, LISTBOX, OLE, PUSHBUTTON, RADIOBUTTON, SCROLLBAR, SPINBOX, TABBOX

Description

Use SetFocus() to make a form or an object in a form ready to receive user input.

When a form has focus, its title bar is highlighted. When an object in a form has focus, its border is highlighted. A form or object with focus is sometimes said to be *current*.

Example

NEW operator syntax:

```
LOCAL f
f=NEW EntryForm()
f.OPEN()
CLASS Entryform OF FORM
  this.View="Company.DBF"
  this.Fld1 = NEW Entryfield(this)
  this.Fld1.Top = 3
  this.Fld1.Left = 1
  this.Fld1.Width=20
  this.Fld1.Datalink = "Company->Company"
  this.Fld2 = NEW Entryfield(this)
  this.Fld2.Top = 5
  this.Fld2.Left = 1
  this.Fld2.Datalink = "Company->State_Prov"
  * to shift focus to Fld2 when you move the mouse
  * over the second field:
  this.Fld2.OnMouseMove = {;this.SetFocus()}
ENDCLASS
```

DEFINE object syntax:

```
DEFINE ENTRYFIELD Company OF THIS;
  Property Datalink "Company->Company",;
  Top 3, Left 1, Width 20
DEFINE ENTRYFIELD State OF THIS;
  Property Datalink "Company->State_Prov",;
  Top 5, Left 1,;
  * to shift focus to Fld2 when you move the mouse
  * over the second field:
  OnMouseMove {;this.SETFOCUS()}
```

See Also

ActiveControl, _curobj, Nextobj, SET CUAENTER

ShapeStyle

Determines the shape of a shape object.

Property of class

SHAPE

Data type

Numeric

Default

The default for ShapeStyle is 3 (Circle).

Description

Use ShapeStyle to specify a shape for a shape object.

The shapes you can specify are as follows:

Value	Shape
0	Rectangle with rounded corners
1	Rectangle
2	Ellipse
3	Circle
4	Square with rounded corners
5	Square

For example, if you want to provide a square, colored background for an area on a form, you can create a shape object with a ShapeStyle value of 5, which specifies a square shape. Use the ColorNormal property to specify the color of the shape object.

Example

The following example creates a form and places an elliptical blue object with a bright white border inside the form.

```
MyForms = NEW FORM("Shape Display")
MyShape = NEW SHAPE(MyForm, "OURSHAPE"    &&Name property = "OURSHAPE"
MyShape.ShapeStyle = 2                    && Elliptical shape
MyShape.ColorNormal = "W+/B"              && Bright white border, blue interior
MyForm.Open()
```



The Name property of the new Shape object contains "OURSHAPE".

See Also

ColorNormal, PenStyle, PenWidth

ShortCut

Specifies a key combination that executes the OnClick subroutine of a menu object.

Property of class

MENU

Data type

Character

Default

The default for ShortCut is an empty string.

Description

Use ShortCut to provide a quick way to execute a menu command with the keyboard. For example, if you assign the character string "CTRL+S" to ShortCut, the user can execute the OnClick subroutine by pressing *Ctrl+S* or *Ctrl+s*.

The value you specify with ShortCut is displayed next to the prompt you specify with the Text property.

Example

NEW operator syntax:

```
FileMnt= NEW MENU ITEM(this)
FileMnt.ShortCut = "Alt+F"
```

DEFINE object syntax:

```
DEFINE MENU ITEM FileMnt OF THIS;
PROPERTY ShortCut "Alt+F"
```

See Also

OnClick

ShowDeleted

Determines if the delete box column in a browse object is displayed.

Property of class

BROWSE

Data type

Logical

Default

The default for ShowDeleted is true (.T.).

Description

Use ShowDeleted to display or omit delete boxes at the left of each record in a browse object.

The setting you give to ShowDeleted has an effect only when SET DELETED is OFF. When SET DELETED is ON, the delete boxes are never displayed.

Example

NEW operator syntax:

```
Br1=NEW BROWSE(this)
Br1.Fields="CompCode,Contact"
Br1.ShowDeleted=.T.
```

DEFINE object syntax:

```
DEFINE BROWSE Br1 OF THIS;
    PROPERTY Fields "CompCode, Contact",;
    ShowDeleted .T.
```

See Also

ShowHeading, ShowRecno

ShowHeading

Determines if field name headings are displayed at the top of each column in a browse object.

Property of class

BROWSE

Data type

Logical

Default

The default for ShowHeading is true (.T.).

Description

Use ShowHeading to determine whether the top line in a browse object displays a record or a heading line.

S

Example

NEW operator syntax:

```
Br1 NEW BROWSE(this)
Br1.Fields = "Compcode,Contact"
Br1.Width = 38
Br1.Height = 17
Br1.Showheading = .F.
```

DEFINE object syntax:

```

DEFINE BROWSE Br1 OF THIS;
PROPERTY;
  Fields "CompCode, Contact",;
  Width 38,;
  Height 17,;
  ShowHeading .F.

```

See Also

ShowDeleted, ShowRecno

ShowRecNo

Determines if the record number column in a browse object is displayed.

Property of class

BROWSE

Data type

Logical

Default

The default for ShowRecNo is true (.T.)

Description

Use ShowRecNo to display or hide record numbers at the left of each record in a browse object.

The ShowRecNo setting affects space allocation in the browse object display. For example, setting ShowRecNo to false (.F.) removes the record number column, leaving more room for displaying fields.

Example

NEW operator syntax:

```

Br1=NEW BROWSE(this)
Br1.Fields="CompCode,Contact"
Br1.ShowRecNo=.F.

```

DEFINE object syntax:

```

DEFINE BROWSE Br1 OF THIS;
PROPERTY Fields "CompCode, Contact",;
  ShowRecNo .F.

```

See Also

ShowDeleted, ShowHeading

ShowSpeedTip

Determines if tips about a control on a form appear in a balloon near the control when the mouse rests on those controls. The controls must have tips text defined via the SpeedTip property for tips to appear.

Property of class

FORM

Data type

Logical

Default

The default for ShowSpeedTip is true (.T.).

Description

Use ShowSpeedTip to determine whether tip messages appear for a control on a form. If ShowSpeedTip is .T., and controls have tips defined via the SpeedTip property, the tip will appear when the mouse comes to rest on the control. If ShowSpeedTip is .F., the tips will not appear. ShowSpeedTip has no effect on controls where no tip has been defined.

Example

In the following example, the tips won't be displayed because ShowSpeedTip has been set to false.

```
f = NEW Form()
f.ShowSpeedTip = .F.
DEFINE Pushbutton PSpeedTip OF f;
Property;
    SpeedTip    "Push to close",;
    OnClick {; Form.Close();};
    Text       "&Close" f.Open()
```

See Also

SpeedTip StatusMessage

Size

Contains the number of elements in an array object.

Property of class

ARRAY

Data type

Numeric

Description

Use Size to find out how many elements an array object has.

Knowing the number of elements in an array lets you execute a FOR...NEXT loop to traverse the array and either read its elements or insert values into them. For more information on array traversal, see Dir().

Example

See Dir() for an example of using Size with array objects.

See Also

ALen(), Delete(), Dimensions, Grow(), Insert(), Resize()

Sizeable

Determines if the user can resize a form.

Property of class

FORM

Data type

Logical

Default

The default for Sizeable is true (.T.).

Description

Set the Sizeable property to false to prevent the user from resizing a form.

When you set Sizeable to true, the form is bordered by a bold line, indicating that the user can change its size and dimensions with the mouse. When you set Sizeable to false (.F.), the form is bordered by a thin line. The pointer does not change when the user places it on the border, and the form can't be resized.

Note When you set the MDI property of a form to true, the Sizeable setting is ignored and the user can always resize the form.

Example

NEW operator syntax:

```
LOCAL F1
F1=NEW Trips()
F1.OPEN()
CLASS Trips OF FORM
    this.MDI = .F.
    this.Text = "Trip Schedule"
    this.Top = 2
    this.Left = 2
    this.Width = 38
    this.Height = 18
    this.Sizeable = .F.
ENDCLASS
```

DEFINE object syntax:

```
DEFINE FORM Trips ;
    PROPERTY Text "Trip Schedule",;
    Top 2, Left 2, Width 38, Height 18,;
    Sizeable .F., MDI .F.
```

See Also

Moveable, Maximize, Minimize

Sort()

Sorts the elements in a one-dimensional array object or sorts rows in a two-dimensional array object.

Property of class

ARRAY

Description

Use the Sort() method to arrange array object elements in alphabetical, numerical, chronological, or logical order.

The Sort() method is similar to the ASORT() function.

The elements to sort in a one-dimensional array object must be of the same data type, and the elements of the column by which rows are to be sorted in a two-dimensional array object must be of the same data type.

Sort() accepts three parameters:

- *<starting element expN>* is the number of an element at which to start sorting if the array object is one-dimensional, or the number (subscript) of the column on which to sort if the array object is two-dimensional. If you omit *<starting element expN>*, Sort() starts sorting at the first element or column in the array.

S

- *<elements to sort expN>* is the number of elements to sort if the array object is one-dimensional, or the number of rows to sort if the array object is two-dimensional. If you omit *<elements to sort expN>*, `Sort()` sorts the rows starting at the row containing element *<starting element expN>* to the last row. If you specify a value for *<elements to sort expN>*, also need to specify a value for *<starting element expN>*.
- *<sort order expN2>* is the sort order. 0 specifies ascending order (the default), and 1 specifies descending order.

For more information on sorting elements in an array object, see `ASORT()`.

Example

```
USE Animals.DBF
* Initialize an array object
ObjArr=NEW ARRAY(RECCOUNT(),3)
* Fill array with values from Animals.DBF
COPY TO ARRAY ObjArr FIELDS Name, Area, Weight
* Use SORT() to order by the area values in
* column 2 of the array.
ObjArr.Sort(2)
* Display sorted array contents
FOR i=1 TO RECCOUNT()
    ? ObjArr[i,1], ObjArr[i,2], ObjArr[i,3]
NEXT i
```

See Also

`ASORT()`, `Dir()`, `Fields()`, `Scan()`

Sorted

Determines whether the prompts in a list box or a combo box are listed in sorted order or in natural order.

Property of class

COMBOBOX, LISTBOX

Data type

Logical

Default

The default for `Sorted` is `false (.F.)`.

Description

Set `Sorted` to `true (.T.)` when you want the prompts in a list box or a combo box to appear in sorted order (alphabetically, numerically, or chronologically). For example, a list of names is more accessible if it is sorted alphabetically.

The natural order of a list box or a combo box depends on the order in which the prompts are generated. For example, when you specify "FILE *.*" for the DataSource property of a list box, the prompts consist of the file names in the default directory. The prompts are created in the order in which the files are listed in the directory, so they are not necessarily arranged alphabetically when you set Sorted to false.

Sorted is non-operational when the DataSource property of the list box or combo box specifies "FIELD" followed by a field name. In this case, the order of prompts in the list box or combo box depends on the record sequence in the table containing the specified field.

Example

NEW operator syntax:

```
LST1 = NEW LISTBOX(this)
LST1.DataSource = "File"
LST1.Sorted = .T.
```

DEFINE object syntax:

```
DEFINE LISTBOX LST1 OF FORM THIS;
PROPERTY DataSource "File",;
Sorted .T.
```

See Also

Multiple

SpeedBar

Determines whether a pushbutton behaves like a SpeedBar button or a standard pushbutton.

Property of class

PUSHBUTTON

Data type

Logical

Default

The default for SpeedBar is false (.F.).

Description

Set SpeedBar to true (.T.) when you want a pushbutton to behave like a SpeedBar button. A SpeedBar button is not included in the tabbing order of a form; consequently, it can't receive focus with *Tab* or *Shift+Tab*.

In Windows applications, SpeedBar buttons are quick alternatives to menu commands. They are typically used for common operations such as Cut, Copy, and Print.

Example

NEW operator syntax:

```
Advance = NEW PUSHBUTTON(this)
Advance.Onclick = ChngRecno
Advance.Text = "Forward"
Advance.SpeedBar = .T.
```

DEFINE object syntax:

```
DEFINE PUSHBUTTON Advance OF THIS;
PROPERTY Onclick ChngRecno;;
Text "Forward", SpeedBar .T.
```

See Also

DEFINE, OnClick

SpeedTip

Specifies the text that appears when the mouse remains on a control for more than one second.

Property of class

CHECKBOX, ENTRYFIELD, PUSHBUTTON, RADIOBUTTON, SPINBOX

Data type

Character

Default

The default for SpeedTip is an empty string.

Description

Use SpeedTip to create a brief text message which appears in a balloon when the mouse rests on a control. Usually this message gives the user a clue as to the function of the control. To suppress the display of Speed Tips, set the ShowSpeedTip property of the form to false (.F.).

Example

```
f = NEW Form()
DEFINE PUSHBUTTON PSpeedTip OF f;
Property;
SpeedTip "Push to close",;
OnClick {; Form.Close()};;
Text "&Close" f.Open()
```

See Also

ShowSpeedTip, StatusMessage

SpinOnly

Determines if users can enter a value in the text box portion of a spin box.

Property of class

SPINBOX

Data type

Logical

Default

The default for SpinOnly is false (.F.).

Description

A spin box lets users enter values in a text box or select values with arrow buttons. When you set the SpinOnly property to false, the text box is enabled. When you set the SpinOnly property to true (.T.) the text box is disabled, restricting input to the predefined values offered by the arrow buttons.

Example

NEW operator syntax:

```
SP1 = NEW SPINBOX(this)
SP1.DataLink = "Country->GNP"
SP1.SpinOnly = .T.;;
SP1.Height = 2
```

DEFINE object syntax:

```
DEFINE SPINBOX SP1 OF THIS;
PROPERTY;
    DataLink "Country->GNP",;
    SpinOnly .T., Height 2
```

See Also

CLASS SPINBOX

StatusMessage

S

Specifies a message to display on the status bar while an object has focus.

Property of class

BROWSE, CHECKBOX, COMBOBOX, EDITOR, ENTRYFIELD, FORM, LISTBOX, MENU, OLE, PUSHBUTTON, RADIOBUTTON, SCROLLBAR, SPINBOX

Data type

Character

Default

The default for `StatusMessage` is an empty string.

Description

Use `StatusMessage` to provide instructions to the user when the user selects an object.

Example

NEW operator syntax:

```
Exit = NEW PUSHBUTTON(this)
Exit.Onclick = Compute
Exit.Text = "Exit"
Exit.StatusMessage = "Click on Exit to compile;
    results"
* Compute is a FUNCTION that performs
* an action and returns .T. or some value.
```

DEFINE object syntax:

```
DEFINE PUSHBUTTON Exit OF THIS;
    PROPERTY Onclick Compute;;
    Text "Exit",;
    StatusMessage "Click on Exit to compile results"
```

See Also

`SpeedTip`, `Text`, `ValidErrorMsg`

Step

Determines how much a user can increment or decrement a value by clicking an arrow in a spin box.

Property of class

`SPINBOX`

Data type

Numeric

Default

The default for `Step` is 1.

Description

Use `Step` to control the rate at which a user can increase or decrease a numeric or date value. For example, a program that expresses large dollar values only in increments of \$500.00 might give a spin box a `Step` value of 500.

Example

NEW operator syntax:

```
Spin1 = NEW SPINBOX(this)
Spin1.Datalink = "Country->GNP"
Spin1.Top = 2
Spin1.Left = 4
Spin1.Height = 2
Spin1.Step = 10
```

DEFINE object syntax:

```
DEFINE SPINBOX Spin1 OF THIS;
PROPERTY Datalink "Country->GNP",;
Top 2, Left 4, Height 2, Step 10
```

See Also

RangeMax, RangeMin

Style

Specifies which parts of a combo box are usable and which parts are displayed automatically.

Property of class

COMBOBOX

Data type

Numeric

Default

The default for Style is 1 (DropDown).

Description

Use Style to determine how the user selects values in a combo box.

The user selects a value from a combo box by entering initial characters in a text box or by selecting the value directly from the prompt list. The setting you give to Style determines whether the text box is usable and whether the prompt list is displayed automatically.

You can give Style one of three values:

Style value	Description
0 (Simple)	The dropdown list is displayed automatically.
1 (DropDown)	The user has to click the arrow to display the dropdown list.
2 (DropDownList)	The user has to click the arrow to display the dropdown list, and the text box does not accept input.

Subscript()

Example

NEW operator syntax:

```
DEFINE COMBOBOX Cb1 OF THIS
  this.Cb1.Style=1
  this.Cb1.DataSource="FIELD Animals->Name"
  this.Cb1.Top=4
  this.Cb1.Left=6
  this.Width=20
  this.Height=12
```

DEFINE object syntax:

```
DEFINE COMBOBOX Cb1 OF THIS FROM 4,6 TO 26,16;
  PROPERTY;
    DataSource "FIELD Animals->Name",;
    Style 1
```

See Also

Enabled

Subscript()

Returns the row number or the column number of a specified element in an array object.

Property of class

ARRAY

Description

Use the Subscript() method when you know the number of an element in a two-dimensional array object and want to reference the element by using its subscripts. The Subscript() method is similar to the ASUBSCRIPT() function.

Subscript() accepts two parameters:

- *<element expN>* is the element number.
- *<row/column expN>* is a number, either 1 or 2, that determines whether you want to return the row or column subscript of an array object. If *<row/column expN>* is 1, Subscript() returns the number of the row subscript. If *<row/column expN>* is 2, Subscript() returns the number of the column subscript.

To determine both the row and column number of an element in a two-dimensional array object, issue Subscript() twice, once with a value of 1 for *<row/column expN>* and once with a value of 2 for *<row/column expN>*. For example, if the element number is 13, the following returns its subscripts:

```
Subscript(Form.aArray,13,1) && Returns row subscript
Subscript(Form.aArray,13,2) && Returns col subscript
```

In one-dimensional array objects, the number of an element is the same as its subscript, so there is no need to use Subscript(). That is, Subscript(3) returns 3, Subscript(5) returns 5, and so on.

Subscript() is the inverse of Element(), which returns the element number when you specify the subscripts of the element.

Example

See Scan() for an example of using Subscript() with array objects.

See Also

ASUBSCRIPT(), Element()

SysMenu

Determines if a form has a Control menu.

Property of class

FORM

Data type

Logical

Default

The default for SysMenu is true (.T.).

Description

Set SysMenu to true to create a Control menu for a form. A standard Windows feature, the Control menu is accessible from the *Control-menu box*, a button at the upper left corner of a form.

The Control menu provides the following options:

- Restore, which restores the form to its original size after the user maximizes it.
- Move, which lets the user move the form with the arrow keys.
- Size, which lets the user resize the form with the arrow keys.
- Minimize, which reduces the form to an icon.
- Maximize, which enlarges the form.
- Close, which closes the form.

When you open a form with the ReadModal() property or the READMODAL() function, the Control menu contains only the Move and Close options.

It's customary to set SysMenu to false when you create a dialog box in a Windows application.

Note When the MDI property of a form is true, the SysMenu setting is ignored and the form always has the Control menu.

Example

NEW operator syntax:

```

F1=NEW EntryForm()
CLASS EntryForm OF FORM
  this.MDI = .F.
  this.Text = "Entry"
  this.Top = 2
  this.Left = 2
  this.Width = 38
  this.Height = 18
  this.SysMenu = .F.
ENDCLASS

```

DEFINE object syntax:

```

DEFINE FORM EntryForm ;
  Property SysMenu .F., MDI .F.,;
  Top 2, Left 2, Width 38, Height 18

```

See Also

Moveable, Sizeable

TabStop

Determines if the user can select an object by pressing *Tab* or *Shift+Tab*.

Property of class

BROWSE, CHECKBOX, COMBOBOX, EDITOR, ENTRYFIELD, LISTBOX, OLE, PUSHBUTTON, RADIOBUTTON, SCROLLBAR, SPINBOX, TABBOX

Data type

Logical

Default

The default for TabStop is true (.T.).

Description

Set the TabStop property to false (.F.) when you want to remove an object from the tabbing order of the parent form. For example, a form might contain a pushbutton that executes a rarely used utility routine. Setting the TabStop property of the pushbutton to false prevents the pushbutton from being selected with *Tab* or *Shift+Tab*.

Example

NEW operator syntax:

```

Comp = NEW ENTRYFIELD(this)
Comp.Datalink = "Clients->Company"
Comp.TabStop = .F.

```

DEFINE object syntax:

```
DEFINE ENTRYFIELD Comp OF THIS;
  PROPERTY Datalink "Clients->Company",;
  TabStop .F.
```

See Also

Before, _curobj, Group

Terminate()

Terminates a conversation with a DDE server application.

Property of class

DDELINK

Description

Use Terminate() to close a DDE link between dBASE and a server application.

Terminate() stops communication between dBASE and the server application, but doesn't close the server application itself.

When you terminate a DDE link with Terminate(), you can restore it with Reconnect. When you terminate the link with the Release() method, the link can't be restored and you need to create the DDELink object again.

Example

```
PUBLIC LinkObj
LinkObj = NEW DDELINK()
LinkObj.Initiate("QPW", "Demo.WB1")
* Subsequent program operations completed; link
* do longer required
LinkObj.Terminate()
```

See Also

Advise(), Execute(), Initiate(), OnNewValue, Peek(), Poke(), Reconnect(), Server, Timeout, Topic, Unadvise()

Text

Specifies a character string to display in or next to an object.

Property of class

BROWSE, CHECKBOX, FORM, MENU, PUSHBUTTON, RADIOBUTTON, RECTANGLE, TEXT

Data type

Character

Default

The default for Text is a character string dBASE assigns to the object when you create it. For example, dBASE automatically assigns "BUTTON1" to the Text property of the first pushbutton you create.

Description

Use Text to label objects or display command labels in menu items. For example, if a pushbutton closes a form, you might set the Text property to "Close" or "Cancel". To display a character string on a form, create a text object and set its Text property to the desired string.

Use a *pick character* to let the user give focus to an object or select a menu item. To designate a character as a pick character, precede it with an ampersand (&). For example, the following command designates C as a pick character:

```
MyForm.MyMenu.FirstItem.Text = "&Close"
```

The pick character is underlined when the object is displayed.

Example

NEW operator syntax:

```
Exit = NEW PUSHBUTTON(this)
Exit.Text = "E&xit"
Exit.Width = 15
Exit.Onclick = {;Form.Close()}
```

DEFINE object syntax:

```
DEFINE PUSHBUTTON Exit OF THIS AT 11,13 ;
PROPERTY Text "E&xit", Width 15,;
OnClick {;Form.Close()}
```

See Also

StatusMessage

TimeOut

Specifies the amount of time in milliseconds that dBASE waits on a transaction before returning an error.

Property of class

DDELINK

Data type

Numeric

Default

The default for TimeOut is 1000 (1 second).

Description

Use TimeOut to set a limit on the length of time dBASE waits for a DDE transaction to complete successfully.

Errors sometimes occur when dBASE tries to exchange data with a server application or send instructions to it. Each time an attempt is made, dBASE waits for the amount of time you specify (in milliseconds) with TimeOut. When the transaction fails to complete in the allotted time, dBASE generates an error message.

When using DDE over a network, you may need to set TimeOut to a larger value.

Example

```
PUBLIC LinkObj
LinkObj = NEW DDELINK()
LinkObj.Timeout = 30
LinkObj.Initiate("QPW", "Demo.WB1")
```

See Also

Advise(), Execute(), Initiate(), OnNewValue, Peek(), Poke(), Server, Terminate(), Topic, Unadvise()

Toggle

Determines if the user can switch between two display modes in a browse object.

Property of class

BROWSE

Data type

Logical

T

Default

The default for Toggle is True (.T.).

Description

Set Toggle to false (.F.) to restrict users to one display mode in a browse object.

A browse object can display multiple records (Browse mode) or one record at a time (Edit mode). By pressing *F2*, the user can toggle between these modes. The default mode is Browse when you set the Mode property to 0; the default mode is Edit when you set Mode to 1 or 2. When you set Toggle to false, the user is restricted to the default mode.

Example

NEW operator syntax:

```
CompanyBrowse = New BROWSE(this)
CompanyBrowse.Top = 3
CompanyBrowse.Left = 1
CompanyBrowse.Width = 60
CompanyBrowse.Alias = "Company"
CompanyBrowse.Toggle = .F.
```

DEFINE object syntax:

```
DEFINE BROWSE CompanyBrowse OF THIS;
  FROM 3,1 TO 13,40;
  Property;
  Alias "Company",;
  Toggle .F.
```

See Also

Browse, Append, Delete

Top

Specifies a position for the top border of an object relative to its parent form or (if the object has no parent form) to the Windows desktop.

Property of class

BROWSE, CHECKBOX, COMBOBOX, EDITOR, ENTRYFIELD, FORM, IMAGE, LINE, LISTBOX, OLE, PUSHBUTTON, RADIOBUTTON, RECTANGLE, SCROLLBAR, SPINBOX, TABBOX, TEXT

Data type

Numeric

Description

Use the Top property in combination with the Left property to position an object in a form or on the Windows desktop. (The Left property specifies the position of the left border.) Top and Left accept integer or non-integer values.

If you used the FROM or AT clause of the DEFINE command *and* specified a value for the Top property, dBASE positions the object with the Top property.

Each unit of the value you assign to Top is the average height of characters in the active font of the parent form. For example, if you increase the Top property of an entry field by 5.5, the entry field is 5.5 character heights above its previous position.

Example

NEW operator syntax:

```
Exit=NEW PUSHBUTTON(this)
Exit.Top = 5
Exit.Left = 2
Exit.Width = 17
Exit.Text = "Exit Program"
Exit.OnClick = {;Form.Close()}
```

DEFINE object syntax:

```
DEFINE PUSHBUTTON Exit OF Entry;
PROPERTY Top 5, Left 2, Width 17;;
Text "Exit Program", OnClick {;Form.Close()}
```

See Also

Bottom, Height, Left, Right, ScaleFontName, ScaleFontSize, Width

Topic

Holds the name of the server document accessed by a DDELink object or the topic of a DDETopic object.

Property of class

DDELINK, DDETOPIC

Data type

Character

Default

The default for Topic is an empty string.

Description

Use Topic to

- Find the server document name that was passed to the Initiate() method of a DDELink object
- Identify the topic of a DDETopic object

Using Topic with DDELink objects

The Initiate() method of a DDELink object opens a channel of communication (known as a *DDE link*) between dBASE and an external Windows application (known as a *server*) and opens one of the data files used by that application (known as a *server document*).

Initiate() requires two parameters:

- *<server>* is the main executable file of the server application
- *<topic>* is the file name of the server document

The Topic property holds the name you pass through *<topic>*.

Using Topic with DDETopic objects

The topic property of a DDETopic object distinguishes the object from other DDETopic objects. For example, a dBASE server application might create two DDETopic objects, one with a topic of NASDAQ and the other with a topic of AMEX:

```
xServer1 = NEW DDETOPIC("NASDAQ")
xServer2 = NEW DDETOPIC("AMEX")
```

Example

```
PUBLIC LinkObj
LinkObj = NEW DDELINK()
LinkObj.Initiate("QPW", "Demo.WB1")
* Subsequent program operations
IF LinkObj.Server="QPW" .AND. ;
    LinkObj.Topic = "Demo.WB1"
    mValue1=LinkObj.Peek("A:A1")
ENDIF
```

See Also

Advise(), Execute(), Initiate(), OnNewValue, Peek(), Poke(), Server, Terminate(), Timeout, Unadvise()

TopMost

Specifies whether forms display on top of all other forms

Property of class

FORM

Description

Use the TopMost property to determine if a form stays in the foreground while focus transfers to other windows.

For example, when an application displays an image object in its own form, it might be desirable to keep the image visible while the user gives focus to other forms. Assigning a value of true (.T.) to the TopMost property keeps the form in the foreground regardless of which form has focus.

TopMost has an effect only when the MDI property is false (.F.).

Example

The following example displays a form containing an image on top of another form. The form with the image is visible even when the other form has focus.

```
f = new TOPMOSTFORM()
f2= new OTHERFORM()
f.open()
f2.open()
CLASS TOPMOSTFORM OF FORM
    this.Top = 18
    this.PageNo = 1
    this.Width = 50
    this.Text = "Modal TopMost"
    this.TopMost = .T.
    this.MDI = .F.
    this.Height = 15
    this.Left = 90
    DEFINE IMAGE IMAGE1 OF THIS;
        PROPERTY;
            Top 2,;
            PageNo 1,;
            DataSource "FILENAME C:\WINDOWS\LEAVES.BMP",;
            Width 30,;
            Alignment 3,;
            Height 10,;
            Left 11
ENDCLASS
CLASS OTHERFORM OF FORM
    this.Top = 4
    this.PageNo = 1
    this.Width = 87
    this.Text = "Other form"
    this.TopMost = .F.
    this.MDI = .T.
    this.Height = 26
    this.Left = 63
ENDCLASS
```

T

See Also

MDI, WindowState

TrackRight

Determines if the user can select a popup menu item with a right mouse click.

Property of class

POPUP

Date type

Logical

Default

The default for TrackRight is true(.T.).

Description

When TrackRight is true (the default), users can select popup menu items with either the right mouse button or the left mouse button.

Set TrackRight to false if you don't want users to be able to select items from a popup menu with a right mouse click.

Example

```
f = NEW Form()
DEFINE POPUP p OF f;
PROPERTY;
    TrackRight .F.
```

See Also

OnLeftMouseDown, OnRightMouseDown

Unadvise()

Asks the server to stop notifying the client when an item in the server document changes.

Property of class

DDELINK

Description

Use the Unadvise() method to terminate a hot link to an item in a server document. A hot link, which you create with the Advise() method, tells the server to notify dBASE when the item changes.

A server document is a file you open in an external application. For example, a data-exchange program might start a session in Quattro Pro for Windows, open one of its spreadsheet files, and establish a hot link to one of its cells. When the hot link is no longer necessary, remove it with Unadvise().

Unadvise() requires the *<item>* parameter, which identifies the item in the server document. This item can be any single element, such as a field in a table or a cell in a spreadsheet. For example, you can remove a hot link from cell C2 of Page A in a Quattro Pro spreadsheet file by passing the parameter "A:C2".

Example

```
PUBLIC LinkObj
LinkObj = NEW DDELINK()
LinkObj.OnNewValue = Valuehandler;
                && Codeblock or function pointer
LinkObj.Initiate("QPW", "Demo.WB1")
LinkObj.Advise("A:A1");
                && Notified when cell A:A1 changes
* Subsequent program operations completed
LinkObj.UnAdvise("A:A1")
```

See Also

Advise(), Execute(), Initiate(), OnNewValue, Peek(), Poke(), Server, Terminate(), Timeout, Topic

Undo()

Reverses the effects of the last Cut or Paste action.

Property of class

BROWSE, COMBOBOX, EDITOR, ENTRYFIELD, SPINBOX

Description

Use Undo() when the user wants reverse the effects of the last Copy, Cut or Paste action. The action of Undo() is identical to the Undo menu item on the standard Windows Edit menu.

If you have assigned a menubar to the form, you can use the menubar's EditUndoMenu property instead of using the Undo() property of individual objects on the form. For more information, see EditUndoMenu.

Example

See Copy() for an example.

See Also

Copy(), Cut(), EditUndoMenu, Paste()

UpBitmap

Specifies the graphic image to display in a pushbutton when it isn't selected.

Property of class

PUSHBUTTON

Data type

Character

Default

The default for UpBitmap is an empty string.

Description

Use UpBitmap to give visual confirmation that a pushbutton is enabled and the user is not clicking it.

The UpBitmap setting can take one of three forms:

- 1 RESOURCE <resource id> <dll name> specifies a bitmap resource and the DLL file that holds it. (A DLL file is a precompiled library of external routines and resources written in non-dBASE languages such as C and Pascal.) You can obtain the resource ID with a resource editor such as Resource Workshop.
- 2 FILENAME <filename> specifies a bitmap file (which usually has the file-name extension .BMP).
- 3 BINARY <binary field> specifies a binary field containing bitmap images. If you use this option, the parent form must be based on a table or a query containing the field. You specify this table or query with the View property of the form. The image displayed always comes from the current record, so moving from record to record changes the image.

When you specify a character string for the pushbutton with Text and an image with UpBitmap, the image is displayed with the character string.

Note You can select a bitmap file with the Choose Bitmap dialog box. To access the Choose Bitmap dialog box, click the Tool button next to the UpBitmap item in the Inspector.

Example

NEW operator syntax:

```
Advance = NEW PUSHBUTTON(this)
Advance.Onclick = ShowInfo
Advance.Text = "Database"
Advance.UpBitmap = "Resource #1904 Bwcc.dll"
Advance.DownBitmap = "Resource Empty Resource.dll"
Advance.StatusMessage = "Click on 'Database' for;
    country info"
```

DEFINE object syntax:

```

DEFINE PUSHBUTTON Advance OF THIS;
  PROPERTY Onclick ShowInfo;;
  Text "Database",;
  UpBitmap "Resource #1904 Bwcc.dll",;
  DownBitmap "Resource Empty Resource.dll",;
  StatusMessage "Click on 'Database' for ;
    country info"

```

See Also

DEFINE, DisabledBitmap, DownBitmap, Enabled, FocusBitmap

Valid

Specifies a condition that must evaluate to true (.T.) before the user can remove focus from an object.

Property of class

EDITOR, ENTRYFIELD, SPINBOX

Data type

Function pointer or codeblock

Description

Use Valid to validate data. For example, an application might use the Valid property of an entry field to detect and prevent invalid account numbers; when a user enters an incorrect value, focus can't be given to another object until the user corrects the entry.

Like other event properties, the Valid property accepts

- Functions
- Procedures
- Codeblocks

For information on these subroutines, see Chapter 4 in the *Programmer's Guide*. For information on writing event-handling routines, see Chapter 14 in the *Programmer's Guide*.

The subroutine you specify for Valid must return a logical value (true or false).

The Valid property resembles the RangeMax and RangeMin properties. However, RangeMax and RangeMin set only upper and lower limits, while Valid can apply other conditions. For example, an upper or lower limit can't exclude nonconsecutive values such as 2, 7, and 9, so a Valid subroutine is required.

Note You can write a subroutine with the Procedure Editor, a window in which you enter dBASE program code. To access the Procedure Editor, click on the Tool button next to the Valid item in the Inspector.

Example

NEW operator syntax:

```
Date = NEW ENTRYFIELD(this)
Date.Datalink = "Clients->Baldate"
Date.Valid = Valchk
```

DEFINE object syntax:

```
DEFINE ENTRYFIELD Date OF THIS;
PROPERTY Datalink "Clients->Baldate",;
Valid Valchk
```

See Also

RangeMax, RangeMin, RangeRequired, ValidErrorMsg, ValidRequired, When

ValidErrorMsg

Specifies a character string to display on the status bar when the Valid property of an entry field returns false (.F.).

Property of class

ENTRYFIELD, SPINBOX

Data type

Character

Default

The default for ValidErrorMsg is "Invalid input".

Description

Use the ValidErrorMsg property to display instructions or a warning when the user enters an incorrect value.

For example, an application might use the Valid property to detect and prevent invalid account numbers. When a user enters an invalid number, the Valid property returns false (.F.) and the message in ValidErrorMsg is displayed in the status bar.

Example

NEW operator syntax:

```
Date = NEW ENTRYFIELD(this)
Date.Datalink = "Clients->Baldate"
Date.Valid = Valchk
Date.ValidErrorMsg = "Invalid date"
* Valchk is a FUNCTION that performs an
* action and returns .T. or some value.
```

DEFINE object syntax:


```

DEFINE ENTRYFIELD Date OF THIS;
  PROPERTY Datalink "Clients->Baldate",;
  Valid Valchk,;
  ValidErrorMsg "Invalid date"

```

See Also

StatusMessage, Valid, ValidRequired

ValidRequired

Determines if the Valid event property applies to all data or to new data only.

Property of class

ENTRYFIELD, SPINBOX

Data type

Logical

Default

The default for ValidRequired is false (.F.).

Description

Set ValidRequired to true to validate existing data as well as new data. For ValidRequired to take effect, you need to set a validation condition with the Valid event property.

You typically set ValidRequired to true (.T.) when you change a validation condition and need to verify and update existing data. For example, a business might add a digit to its account numbers and change the Valid property of an entry field to require the new digit. If the ValidRequired property is set to true, dBASE also detects any existing account numbers that lack the digit and forces the user to make appropriate changes.

The Valid property resembles the RangeMax and RangeMin properties. However, RangeMax and RangeMin set only upper and lower limits, while Valid can apply other conditions. For example, an upper or lower limit can't exclude nonconsecutive values such as 2, 7, and 9, so a Valid subroutine is required.

Example

NEW operator syntax:

```

Date = NEW ENTRYFIELD(this)
Date.Datalink = "Clients->Baldate"
Date.Valid = Valchk
Date.ValidRequired = .T.
Date.ValidErrorMsg = "Invalid date"
* Valchk is a FUNCTION that performs an
* action and returns .T. or some value

```

DEFINE object syntax:

```

DEFINE ENTRYFIELD Date OF THIS;
  PROPERTY Datalink "Clients->Baldate",;
  Valid Valchk, ValidRequired .T.,;
  ValidErrorMsg "Invalid date"

```

See Also

RangeMax, RangeMin, RangeRequired, Valid, ValidErrorMsg

Value

The value contained in an object.

Property of class

CHECKBOX, COMBOBOX, EDITOR, ENTRYFIELD, LISTBOX, RADIOBUTTON, SCROLLBAR, SPINBOX

Data type

Numeric, float, character, or date

Default

The default for Value varies for each object. For example, the default for an entry field is the value stored in the table field (that the entry field is linked to) for the first record. For a check box, the default is .F. (a check box is unchecked by default).

Description

Use Value to initialize data in an object or query a user's input. For example, you can use the Value property of a spin box to set the default value that appears in the text box. You can also query the Value property of an entry field to obtain the value entered by the user.

Example

NEW operator syntax:

```

BDate = NEW ENTRYFIELD(this)
BDate.Datalink = "Clients->Baldate"
BDate.Value = DATE()

```

DEFINE object syntax:

```

DEFINE ENTRYFIELD BDate OF THIS;
  PROPERTY Datalink "Clients->Baldate",;
  Value DATE()

```

See Also

DataLink, DataSource, LISTSELECTED(), Selected(), STORE

Vertical

Determines whether a scroll bar is vertical or horizontal.

Property of class

SCROLLBAR

Data type

Logical

Default

The default for Vertical is true (.T.).

Description

Use Vertical to make a scroll bar fit with other objects on a form or to enhance the design of the form. For example, a form might not have space for a horizontal scroll bar, or a vertical scrollbar might look awkward next to a spin box.

Example

NEW operator syntax:

```
IncNum = NEW SCROLLBAR(this)
IncNum.Top = 7
IncNum.Left = 3
IncNum.DataLink = "Num"
IncNum.Vertical = .F.
```

DEFINE object syntax:

```
DEFINE SCROLLBAR IncNum OF THIS;
  PROPERTY DataLink "Num", Vertical .F.,;
  Top 7, Left 3
```

See Also

DataLink, Height, Left, Top, Width

View

Specifies the name of a query or a table on which a form is based.

Property of class

FORM

Data type

Character

U-V

Default

The default for View is an empty string.

Description

Use View to determine which tables are automatically opened whenever the form is opened. You can assign multiple tables with a query (.QBE, .VUE), or you can assign a single table (.DBF). Although you're not required to set the View property, you should if the form accesses table data. Otherwise, every time you open the form, you need to open the table or query first.

Once you associate the form with a table or tables, you need to link objects such as entry fields and check boxes with specific fields in the table.

If you want certain tables open while you design a form, but may not want the same tables open at runtime, use DesignView instead of View. For more information, see DesignView.

Notes You can specify a value for the View property with the Choose View dialog box, which lets you choose a query or a table. To access the Choose View dialog box, click on the Tool button next to the View item in the Inspector.

Example

NEW operator syntax:

```
LOCAL
F1=NEW EntryForm()
CLASS EntryForm OF FORM
this.Text = "Entry Form"
this.Top = 2
this.Left = 2
this.Width = 38
this.Height = 18
this.View = "Company94.QBE"
* Company94.QBE is a Query file name on which
* the Form Entry is based.
ENDCLASS
```

DEFINE object syntax:

```
DEFINE FORM EntryForm;
Property Text "Entry Form",;
View "Company94.QBE",;
Top 2, Left 2, Width 38, Height 18
```

See Also

Alias, DataLink, DataSource, DesignView

Visible

Determines whether an object is visible or hidden.

Property of class

BROWSE, CHECKBOX, COMBOBOX, EDITOR, ENTRYFIELD, FORM, IMAGE, LINE, LISTBOX, OLE, PUSHBUTTON, RADIOBUTTON, RECTANGLE, SCROLLBAR, SPINBOX, TABBOX, TEXT

Data type

Logical

Default

The default for Visible is true (.T.).

Description

Use Visible when you want to display an object only if a specified condition is met or a specified action is taken. For example, you can specify conditions with event properties (such as OnClick and Valid) that set Visible to true when the user clicks an object or enters certain values.

Example

NEW operator syntax:

```
Spin1 = NEW SPINBOX(this)
Spin1.Datalink = "Value1"
Spin1.Top = 2
Spin1.Left = 4
Spin1.Height = 2
Spin1.Visible = .F.
```

DEFINE object syntax:

```
DEFINE SPINBOX Spin1 OF THIS;
PROPERTY Datalink "Value1",;
Top 2, Left 4, Height 2, Visible .F.
```

See Also

Enabled

When

Specifies a condition that must evaluate to true (.T.) before the user can give focus to an object.

Property of class

BROWSE, CHECKBOX, COMBOBOX, EDITOR, ENTRYFIELD, LISTBOX, PUSHBUTTON, RADIOBUTTON, SCROLLBAR, SPINBOX, TABBOX

Data type

Function pointer or codeblock

Description

Use When to determine when an object is available to users. For example, an application that manages sensitive customer accounts might prevent a browse object from receiving focus until the user enters the proper password into an entry field.

Like other event properties, the When property accepts

- Functions
- Procedures
- Codeblocks

For information on these subroutines, see Chapter 4 in the *Programmer's Guide*. For information on writing event-handling code, see Chapter 14 in the *Programmer's Guide*.

The subroutine you specify for When must return a logical value (true or false).

Note You can write a subroutine with the Procedure Editor, a window in which you enter dBASE program code. To access the Procedure Editor, click on the Tool button next to the When item in the Inspector.

Example

NEW operator syntax:

```
USE Company
* Form declaration
NextRec = NEW PUSHBUTTON(this)
NextRec.Text = "Next"
NextRec.When = {;.NOT. EOF()}
NextRec.OnClick = {;SKIP}
```

DEFINE object syntax:

```
DEFINE PUSHBUTTON NextRec OF THIS;
PROPERTY;
Text "Next",;
When {;.NOT. EOF()},;
OnClick {;SKIP}
```

See Also

Valid

Width

Specifies the width of an object.

Property of class

BROWSE, CHECKBOX, COMBOBOX, EDITOR, ENTRYFIELD, FORM, IMAGE, LINE, LISTBOX, OLE, PUSHBUTTON, RADIOBUTTON, RECTANGLE, SCROLLBAR, SPINBOX, TABBOX, TEXT

Data type

Numeric

Description

Use the Width property in combination with the Height property to adjust the size of an object.

Width determines the distance from the left border to its right border. You can assign an integer or non-integer value to Width.

Each unit of the value you assign to Width is the average width of characters in the active font of the parent form. For example, if you set the Width property of an entry field to 15.5, the entry field is as wide as 15.5 characters.

Example

NEW operator syntax:

```
Action1 = NEW PUSHBUTTON(this)
Action1.Text = "Search for Open Accounts"
Action1.Onclick = Search
Action1.Width = 22
Action1.Top = 11
Action1.Left = 13
```

DEFINE object syntax:

```
DEFINE PUSHBUTTON Action1 OF THIS ;
PROPERTY Text "Search for Open Accounts",;
Top 11, Left 13, Width 22
OnClick Search
```

See Also

Left, Height, ScaleFontName, ScaleFontSize, Top

WindowMenu

Specifies a menu object that displays a list of all open MDI windows.

Property of Class

MENUBAR

Data type

Object reference

Description

WindowMenu contains a reference to a menu object that has a menubar as its parent. When users open this menu object, dBASE displays a pulldown list of all open MDI windows.

WindowMenu automatically places a separator line on the pulldown list between any menu prompts and the list of open windows. The currently active window shows a check next to the window name.

If you use the Menu Designer to create a menubar, WindowMenu is automatically set to an item named Window on the menubar:

```
this.WindowMenu = this.Window
```

Example

```
NEW SAMPLEMENU(FormObj,"Root")
CLASS SAMPLEMENU(FormObj,Name) OF MENUBAR(FormObj,Name)
  DEFINE MENU FILE OF THIS;
  PROPERTY;
  Text "&File"
  DEFINE MENU EXIT OF THIS.FILE;
  PROPERTY;
  Text "E&xit"
  DEFINE MENU EDIT OF THIS;
  PROPERTY;
  Text "&Edit"
  DEFINE MENU UNDO OF THIS.EDIT;
  PROPERTY;
  Text "&Undo"
  DEFINE MENU CUT OF THIS.EDIT;
  PROPERTY;
  Text "Cu&t"
  DEFINE MENU COPY OF THIS.EDIT;
  PROPERTY;
  Text "&Copy"
  DEFINE MENU PASTE OF THIS.EDIT;
  PROPERTY;
  Text "&Paste"
  DEFINE MENU WINDOW OF THIS;
  PROPERTY;
  Text "&Window"
```



```

        DEFINE MENU ARRANGE OF THIS.WINDOW;
        PROPERTY;
        Text "&Arrange"
    DEFINE MENU HELP OF THIS;
    PROPERTY;
    Text "&Help"
    DEFINE MENU ABOUT OF THIS.HELP;
    PROPERTY;
    Text "&About"
    This.EditUndoMenu = This.Edit.Undo
    This.EditCutMenu = This.Edit.Cut
    This.EditCopyMenu = This.Edit.Copy
    This.EditPasteMenu = This.Edit.Paste
    This.WindowMenu = This.Window
ENDCLASS

```

See Also

CLASS MENUBAR, EditCopyMenu, MDI

WindowState

Determines if a form is minimized, maximized, or displayed in its original size.

Property of class

FORM

Data type

Numeric

Default

The default for WindowState is 0 (Normal).

Description

Use WindowState to maximize, minimize, or restore a form.

The WindowState property accepts the following values:

Setting	Effect on window
0 (Normal)	Returns the form to its original size
1 (Minimized)	Reduces the form to an icon
2 (Maximized)	Enlarges the form to cover the work area of the desktop

When you set the WindowState property to 0, the form is said to be *restored*.

Example

NEW operator syntax:

```
LOCAL f
f=NEW Trips()
f.OPEN()
CLASS Trips OF FORM
  this.Text = "Trip Schedule"
  this.Top = 2
  this.Left = 2
  this.Width = 38
  this.Height = 18
  this.WindowState = 2
ENDCLASS
```

DEFINE object syntax:

```
DEFINE FORM Trips FROM 2,2 TO 20,40;
  PROPERTY Text "Trip Schedule",;
  Top 2, Left 2, Width 38, Height 18.;
  WindowState 2
OPEN FORM TRIPS
```

See Also

Moveable, Sizeable

Wrap

Determines if an editor object wraps input text automatically.

Property of class

EDITOR

Data type

Logical

Default

The default for Wrap is true (.T.).

Description

Set Wrap to false (.F.) when you want the user to control the width of each line of text. Set Wrap to true (the default) if you want dBASE to execute a carriage return automatically when a text string exceeds the width of an editor object.

Example**NEW operator syntax:**

```
Ed1=NEW EDITOR(this)
Ed1.Wrap = .T.
Ed1.Top = 4
Ed1.Left = 37
Ed1.Height = 12
Ed1, Width = 32
Ed1.DataSource = "MEMO Contact->Notes"
```

DEFINE object syntax:

```
DEFINE EDITOR Ed1 OF THIS;
PROPERTY Top 4, Left 37, Height 12, Width 32,;
Wrap .T.,;
DataSource "MEMO Contact->Notes"
```

See Also

Width, Height



SQL

Local SQL

Visual dBASE provides the ability to mix dBASE and SQL commands for operations against both local and remote data. This chapter describes the syntax of SQL commands that can be used within dBASE when working with non-database server data (i.e. dBASE and Paradox tables).

Note that the syntax described here is for use against dBASE/Paradox tables only. Database servers (such as Interbase or Oracle) have their own implementations of SQL syntax. When working with server data, the server's syntax must be used. For detailed information on SQL support in *Visual* dBASE, see the *Programmer's Guide*. For information on the SQL dialect and extensions used at your server, see your SQL server documentation.

Memory variable substitution in SQL queries

dBASE supports the substitution of memvar values in SQL queries. dBASE memory variables are indicated with a colon, as in the following example

```
x = "Robert"
SELECT * FROM customers WHERE firstname = :x
```

The memory variable `x` is resolved and "Robert" is substituted in its place.

Naming conventions

Table names

Table names may be comprised of alphanumeric characters, underscores (`_`), and the period (`.`). They may include full file and path specifications or BDE alias specifications (in the format `:ALIASNAME:TABLENAME`). They may even duplicate SQL keywords.

However, table names which include anything other than alphanumeric characters and underscores, or include file or alias specifications, must always be enclosed in single or double quotes. For example:

ALTER TABLE

SELECT * FROM 'C:\SAMPLE.DAT\TABLE'	includes full path specification
SELECT * FROM "TABLE.DBF"	includes a period
SELECT PASSID FROM "PASSWORD"	duplicates an SQL keyword
SELECT BID_DATE FROM ":FSFDBASE:BIDS"	includes BDE alias specification

Column names

Column names may be comprised of alphanumeric characters, underscores (_), and spaces (.). They may also duplicate SQL keywords.

However, column names that include spaces or duplicate SQL keywords must always be:

- enclosed in single or double quotes
- prefaced with an SQL table name or table correlation name, in the format
TABLENAME.COLUMNNAME.

For example:

SELECT E."EMP ID" FROM EMPLOYEE	includes a space
SELECT DATELOG."DATE" FROM TABLE	duplicates an SQL keyword

ALTER TABLE

Adds or drops (deletes) one or more columns (fields) from a table.

Syntax

ALTER TABLE <table name> ADD <column name><data type> | DROP <column name>
[, ADD <column name><data type> ...] [, DROP <column name>...]

Description

Use ALTER TABLE to modify the structure of an existing table. ALTER TABLE with the ADD clause adds the column <column name> of the type <data type> to <table name>. Use the DROP clause to remove the existing column <column name> from <table>.

Warning Data stored in a dropped column is lost without warning, regardless of the SET SAFETY setting.

Multiple columns may be added and/or dropped in a single ALTER TABLE command.

Use ALTER TABLE as a means of modifying the structure of a table without using the dBASE Table Structure dialog.

Example

The following statement adds two columns (REFER and LASTCALL) and deletes one column (FIRST_CONT) from a the table CUSTOMER.DBF:

```
ALTER TABLE CUSTOMER ADD REFER CHAR(20), ADD LASTCALL DATE, DROP FIRST_CONT
```

See also

CREATE TABLE, DROP TABLE, INSERT, MODIFY STRUCTURE (dBASE)

CREATE INDEX

Creates a new index on a table.

Syntax

```
CREATE INDEX <index name> ON <table name> <column name> [, <column name>...]
```

Description

Use CREATE INDEX to create a new index *<index name>*, in ascending order, based on the values in one or more columns *<column name>* of *<table name>*. Unlike the dBASE language, expressions cannot be used to create an index, only columns.

When working with dBASE .DBF tables, the index can only be created for a single column. The new index is created as a new index tag in the production index. A production index is created if it does not exist.

CREATE INDEX is equivalent to the INDEX ON *<field list>* TAG *<tag name>* syntax in the dBASE language.

Example

The following statement adds an index called ZIP on the ZIP_POSTAL column of the CUSTOMER.DBF table:

```
CREATE INDEX ZIP ON CUSTOMER ZIP_POSTAL
```

See also

DROP INDEX, INDEX (dBASE)

CREATE TABLE

Creates a new table.

Syntax

```
CREATE TABLE tablename (<column name> <data type> [,<column name> <data type>...])
```

Usage

Use CREATE TABLE to create a new table. The type of table produced (dBASE or Paradox) depends on the current setting of SET DBTYPE.

At least one *<column name> <data type>* must be defined. The column definition list must be enclosed in parentheses.

The following table lists SQL syntax for data types used with CREATE TABLE, and describes how they are mapped to dBASE and Paradox types.

Table 9.1 Data type mappings for CREATE¹

SQL Syntax	dBASE	Paradox
SMALLINT	Numeric	Short
INT	Numeric	Long
DECIMAL(x,y)	N/A	BCD
NUMERIC(x,y)	Numeric(x,y)	Number
FLOAT(x,y)	Numeric(x,y)	Number
CHARACTER(n)	Character	Alpha
DATE	Date	Date
BOOLEAN	Logical	Logical
BLOB(n,s) ²	Memo/Binary	Memo/Binary
TIME	N/A	Time
TIMESTAMP	N/A	TimeStamp
MONEY	Numeric(20,4)	Money
AUTOINC	N/A	Autoincrement
BYTES(n)	N/A	Bytes

- 1. Parameters
x = length; if omitted, to 20 for dBASE
y = decimal places; if omitted, defaults to 4 for dBASE
n = length; if omitted, length defaults to 1
s = subtype; if a blob subtype is omitted, the subtype defaults to Memo
- 2. Blob subtypes
BINARY
FMTMEMO (Paradox only)
GRAPHIC (Paradox only)
MEMO
OLE

CREATE TABLE is a alternate way of creating a table without using the dBASE Table Structure dialog or the dBASE CREATE STRUCTURE EXTENDED, CREATE FROM commands.

Examples

The following example creates a dBASE table called SALES with the following structure:

Table 9.2 SALES.DBF structure

Field name	Field type	Field length	Decimal places
SALESID	Character	6	
CUSTOMERID	Character	10	
ORDERDATE	Date	8	
ORDERNMBR	Numeric	7	0

Table 9.2 SALES.DBF structure (continued)

Field name	Field type	Field length	Decimal places
ORDERAMT	Numeric	9	2
DELIVERED	Logical	1	

```
CREATE TABLE SALES (
    SALESID      CHAR(6),
    CUSTOMERID  CHAR(10),
    ORDERDATE    DATE,
    ORDERNMBR   NUMERIC(7,0),
    ORDERAMT    NUMERIC(9,2),
    DELIVERED    BOOLEAN)
```

See also

ALTER TABLE, CREATE (dBASE), CREATE FROM (dBASE), CREATE STRUCTURE EXTENDED (dBASE), DROP TABLE

DELETE FROM

Deletes rows (records) from a table.

Syntax

```
DELETE FROM <table name> [WHERE <search condition>]
```

Usage

Use DELETE FROM to delete rows, or records, from <table name>. Without the WHERE clause, all the rows in the table are deleted. Use the WHERE clause to specify a <search condition>. Only records matching the <search condition> are deleted.

When DELETE FROM is run against dBASE .DBF tables the following rules apply:

- 1 If a WHERE clause is used, DELETE FROM only **marks** rows for deletion, even if all the rows match the <search condition>. In this way, DELETE FROM behaves like the dBASE DELETE command. The rows are recallable unless the table is packed.
- 2 Without the WHERE clause, all the rows in the table are actually deleted. In this case, DELETE FROM behaves like the dBASE ZAP command. The rows are not recallable, and the table will have zero rows.

When DELETE FROM is run against a Paradox table, all the rows matching the <search condition> are actually deleted. If no WHERE clause is used, all the rows in the table are deleted. The data in the deleted rows is **not** recallable.

Example

The following example deletes all the rows in a dBASE table called CUSTOMER and results in a table with zero rows.

```
DELETE FROM CUSTOMER
```

The following example marks all the rows in a dBASE table called CUSTOMER for deletion, but does not actually delete the rows from the table.

```
DELETE FROM CUSTOMER WHERE CUSTOMER_N > 0
```

The following example marks all the rows where the CITY field is equal to “Freeport” for deletion in a dBASE table called CUSTOMER.

```
DELETE FROM CUSTOMER WHERE CITY = "Freeport"
```

The following example deletes all the rows where the CITY field is equal to “Freeport” in a Paradox table called CUSTOMER.

```
DELETE FROM "CUSTOMER.DB" WHERE CITY = "Freeport"
```

See also

DELETE (dBASE), PACK (dBASE), SELECT, ZAP (dBASE)

DROP INDEX

Drops (deletes) an existing index from a table.

Syntax

```
DROP INDEX <table name>.<index name>
```

Usage

Use DROP INDEX to drop, or delete, the index <index name> from <table name>. For dBASE .DBF tables <index name> must be the name of a tag in the production index.

Example

The following statement drops the index tag NAME from the production index of a dBASE table called EMPLOYEE:

```
DROP INDEX EMPLOYEE.NAME
```

The following statement drops a primary index on the Paradox table, EMPLOYEE.DB:

```
DROP INDEX "EMPLOYEE.DB".PRIMARY
```

See also

CREATE INDEX, DELETE TAG (dBASE), DROP TABLE

DROP TABLE

Drops (deletes) a table.

Syntax

```
DROP TABLE <table name>
```

Usage

Use DROP TABLE to delete the table *<table name>* from disk. The associated production index file and memo file, if any, are also deleted.

Example

The following statement drops a dBASE table call EMPLOYEE:

```
DROP TABLE EMPLOYEE
```

See also

CREATE TABLE, DELETE FROM, DELETE TABLE (dBASE)

INSERT INTO

Adds new rows (records) to a table.

Syntax

```
INSERT INTO <table name> [(<column list>)] VALUES (<value list>) | SELECT <command>
```

Usage

Use INSERT INTO to add rows, or records, to a table. There are two forms of this command. In the first form, you use *<value list>* to specify individual column values that are to be inserted for the new row. The values to be inserted must match in number, order, and type with the columns specified in *<column list>*, if *<column list>* is specified. Columns in the new row for which no value is given are left blank. If no *<column list>* is given, the order of the columns as they appear in the table is assumed. Without a *<column list>* a value must be provided for each column in the *<value list>*.

In the second form, the SELECT clause is executed just like a SELECT command. The row or rows returned by the SELECT are inserted into *<table name>*. The columns of the rows returned by the SELECT are matched up with the columns listed in *<column list>*. Therefore, the columns returned by SELECT must match in number, order, and type with the columns specified in *<column list>*, if *<column list>* is specified. If no *<column list>* is given, the number, order, and type of the columns returned by the SELECT must match the number, order, and type of the columns in *<table name>*.

Example

The following example makes a copy of the structure of the dBASE table CUSTOMER.DBF called CACUST.DBF, then adds customers to the new file.

```
USE CUSTOMER                && open the customer table
COPY STRUCTURE TO CACUST    && copy the structure
USE                          && close customer
* The next line adds a new row (record) to CACUST and inserts John Smith, Riverside, CA
* in the NAME, CITY, and STATE_PROV columns (fields) respectively
INSERT INTO CACUST (NAME, CITY, STATE_PROV) VALUES ("John Smith", "Riverside", "CA")
* The next line retrieves all the records where the state is CA from CUSTOMER and adds
them to CACUST
INSERT INTO CACUST SELECT * FROM CUSTOMER WHERE STATE_PROV = "CA"
```

See also

APPEND (dBASE), APPEND BLANK (dBASE), APPEND FROM (dBASE), COPY (dBASE), COPY TABLE (dBASE), CREATE TABLE, REPLACE (dBASE), SELECT

SELECT

Retrieves data from one or more tables.

Syntax

```
SELECT <column list> FROM <table list> [WHERE <search condition>] [GROUP BY <column list>]
[ORDER BY <column list>] [HAVING <search condition>] [SAVE TO <filename>]
[ALIAS <alias name>]
```

Usage

Use SELECT to retrieve data from a table or set of tables based on some criteria.

The *<column list>* is a comma-delimited list of columns in the table(s) that you wish to retrieve. The columns are retrieved in the order given in the list. If two or more tables used by SELECT use the same field names, distinguish the tables by using the table name and a dot ("."). For example, if you're SELECTing from the CUSTOMER table and the PRODUCT table, and they both have a field called NAME, enter the fields as CUSTOMER.NAME and PRODUCT.NAME in *<column list>*. To retrieve all the columns from *<table list>*, use an asterisk (*) for *<column list>*. To eliminate rows containing duplicate values within the same column, precede the *<column list>* with the keyword DISTINCT.

FROM <table list> The FROM clause specifies the table or tables from which to retrieve data. *<table list>* can be a single table or a comma-delimited list of tables.

WHERE <search condition> The optional WHERE clause reduces the number of rows returned by a SELECT to those that match the criteria specified in *<search condition>*.

GROUP BY <column list> The optional GROUP BY clause specifies how retrieved rows are grouped for aggregate functions. Any column names that appear in the GROUP BY *<column list>* must also appear in the SELECT *<column list>*.

ORDER BY <column list> The optional ORDER BY clause specifies the column to order the retrieved rows by.

HAVING <search condition> The optional HAVING clause specifies a *<search condition>* that evaluates to being true or false for each row in the group.

SAVE TO <filename> The optional SAVE TO clause specifies that the results of the SELECT are to be saved to a new table called *<filename>*.

ALIAS <alias name> The optional ALIAS clause specifies the name of the alias given to the work area the results of the SELECT appear in. If not specified, ALIAS defaults to SQL_*<integer>*.

The default output of SELECT statements produces an open workarea (similar to USE). All dBASE commands and functions can be used on the result of a SELECT. If the query produced a temporary table, the temporary table is deleted when the answer set is

closed. If you wish to save the results, use the SAVE TO option of SELECT or use the COPY TO command after the SELECT.

SELECT opens the answer set in the first unused workarea, starting from area 1. If the query was successful, the currently selected area is changed to the workarea where the answer set was produced.

Examples

The following examples show simple SELECTs:

```
SELECT NAME, PHONE FROM CUSTOMER WHERE STATE_PROV = "CA"
SELECT CUSTOMER_NO FROM CUSTOMER WHERE LAST_NAME = "Johnson"
SELECT PART_NO, SUM(QUANTITY) AS PQTY FROM PARTS GROUP BY PART_NO
```

The following example shows a join in which fields from each table are involved in some type of equality check require a WHERE clause:

```
SELECT DISTINCT PARTS.PART_NO, PARTS.QUANTITY, GOODS.CITY FROM PARTS, GOODS
WHERE PARTS.PART_NO = GOODS.PART_NO AND PARTS.QUANTITY > 20
ORDER BY PARTS.QUANTITY, GOODS.CITY, PARTS.PART_NO
```

The following example shows the use of the DESCENDING keyword in the ORDER BY clause. Note that in this case you must also specify DISTINCT.

```
SELECT DISTINCT CUSTOMER_NO FROM CUSTOMER ORDER BY CUSTOMER_NO DESCENDING
```

See also

CREATE QUERY(dBASE), DELETE FROM, INSERT FROM, SET FILTER (dBASE), SET KEY (dBASE), SET RELATION (dBASE), UPDATE

UPDATE

Adds or changes values in existing columns in existing rows of a table.

Syntax

```
UPDATE <table name> SET <column name> = <expression> [, <column name> = <expression>...]
WHERE <search condition>
```

Usage

Use UPDATE to update (change) values within existing columns in existing rows of a table. The column specified by <column name> is updated with the value of <expression> in all rows that match the <search criteria> of the WHERE clause. If the WHERE clause is omitted, the column is updated in all rows in the table. Multiple columns may be updated in a single UPDATE command. A given column of a table may only appear once to the left of a "=" in the SET clause.

Example

The following command updates that YTD sales to zero for each customer that was contacted in the previous calendar year:

```
UPDATE CUSTOMER SET YTD_SALES = 0 WHERE FIRST_CONT < {01/01/95}
```

See also

INSERT FROM, REPLACE (dBASE) SELECT



Appendixes



Changes since dBASE IV 2.0

This appendix describes changes in the dBASE language from dBASE IV version 2.0 to *Visual* dBASE. It lists the new, changed, and unsupported language elements.

New language elements

Element	Description
ACOPY()	Copies elements from one array to another.
ADEL()	Deletes an element, row, or column from an array.
ADIR()	Stores file characteristics of files in a directory (name, size, date stamp, time stamp, and DOS attributes) to an array.
AELEMENT()	Returns the number of a specified element in a one- or two-dimensional array.
AFIELDS()	Stores structural information about a table in an array.
AFILL()	Inserts a specified value into one or more elements of an array.
AGROW()	Adds an element, row, or column to an array.
AINS()	Inserts .F. values into an array.
ALEN()	Returns the number of elements, rows, or columns of an array.
ANSI()	Returns the ANSI value that corresponds to a specified OEM (DOS code page) character expression.
ARESIZE()	Increases or decreases the array size.
ASCAN()	Searches an array for an expression, returning the number of the first element that matches the expression.
ASORT()	Sorts the elements in an array.
ASUBSCRIPT()	Returns the row or column number of a specified element in an array.
BEGINTRANS()	Replaces BEGIN TRANSACTION and END TRANSACTION.
BINTYPE()	Returns the predefined type number of a specified binary field.

New language elements (continued)

Element	Description
BITAND()	Performs an AND comparison of the bits in two specified expressions.
BITLSHIFT()	Returns a number generated by left-shifting the bits in a specified expression.
BITRSHIFT()	Returns a number generated by right-shifting the bits in a specified expression.
BITOR()	Performs an OR comparison of the bits in two specified expressions.
BITSET()	Indicates whether a bit in a specified expression is set.
BITXOR()	Performs an exclusive OR (XOR) comparison of the bits in two specified expressions.
BOOKMARK()	Returns a bookmark (similar to a record pointer) for the current record of a dBASE, Paradox, or SQL table.
BUILD	Links object code files (.PRO, .WFO) and resources into a Windows executable file (.EXE) if the optional <i>Visual</i> dBASE Compiler is installed.
CD	Changes the current default drive or directory.
CENTER()	Returns a character string that contains a string centered in a line of specified length.
CHARSET()	Returns the name of the character set that the current or specified table is using.
CHOOSEPRINTER()	Lets you choose a printer or specify print options, also resetting the appropriate system memory variables.
CLASS...ENDCLASS	Declares a class of objects and optionally specifies the member properties and methods for that class.
CLEAR AUTOMEM	Initializes a set of automem variables corresponding to fields in the current table.
CLEAR PROGRAM	Clears from memory all compiled program files that aren't currently executing and aren't currently open with SET FORMAT, SET PROCEDURE, or SET LIBRARY.
CLOSE FORMS	Clears forms from the screen without releasing their definitions from memory.
CLOSE TABLES	Closes tables in all work areas, or close all tables in the current database, if one is selected.
COMMIT()	Ends a transaction and writes to the open files any changes made during the transaction.
COPY BINARY	Copies the contents of the specified binary field to a file.
COPY TABLE	Copies a specified table to a file.
CREATE CATALOG	Creates a new catalog.
CREATE COMMAND	Displays a specified program file for editing, or displays an empty editing window.
CREATE FILE	Displays a specified text file for editing, or displays an empty editing window.
CREATE FORM	Opens the Form Designer to create or modify a form (.WFM) file.
CREATE MENU	Opens the Menu Designer to create or modify a menu (.MNU) file.
CREATE SESSION	Creates a new session, which lets you open the same table again as if you were working in a multiuser environment.

New language elements (continued)

Element	Description
CREATE...STRUCTURE EXTENDED	Creates and opens a table that you can use as a template for a new table.
DATABASE()	Returns the name of the current database from which tables are accessed.
DBERROR()	Returns the number of the last IDAPI error.
DBMESSAGE()	Returns the error message of the last or a specified IDAPI error.
DEFINE	Creates an object from a built-in or custom class.
DEFINE COLOR	Creates and names a customized color.
DELETE TABLE	Deletes a specified table.
DISPLAY/LIST COVERAGE	Displays the contents of a coverage file (.COV).
DO...UNTIL	Executes the statements between DO and UNTIL as long as a specified condition is false or until dBASE encounters an EXIT command.
DOS	Temporarily passes control to the DOS prompt, allowing execution of DOS commands.
ELAPSED()	Returns the number of seconds elapsed between two specified times.
EMPTY()	Returns .T. if a specified expression or field is blank, .F. if it contains data.
EXTERN	Declares a prototype for a non-dBASE function contained in a DLL file. A prototype tells dBASE to convert its arguments to data types the external function can use, and to convert the value returned by the external function into a data type dBASE can use.
FACCESSDATE()	Returns the last date a file was opened under Windows 95.
FCREATEDATE()	Returns the date a file was created under Windows 95.
FCREATETIME()	Returns the time a file was created under Windows 95.
FDECIMAL()	Returns the number of decimal places in a specified field of a table.
FLENGTH()	Returns the length of the field in a specified position of a table.
FNAMEMAX()	Returns the maximum allowable file-name length on a given drive or volume.
FOR...NEXT	Executes the statements between FOR and NEXT the number of times indicated by the FOR statement.
FSHORTNAME()	Returns the short name (i.e. the DOS compatible name) of a file created under Windows 95.
FUNIQUE()	Creates a unique file name.
GENERATE	Adds records containing random data to the current table.
GETCOLOR()	Lets you define a custom color or select a color from the color palette.
GETDIRECTORY()	Displays a dialog box from which you can select a directory for use with subsequent commands.
GETEXPR()	Displays the Create an Expression or the Edit an Expression dialog box, and returns the expression you specify.
GETFILE()	Displays a dialog box from which you can choose a file name.
GETFONT()	Calls a dialog box from which you can select character fonts.
HTOI()	Returns the decimal-number equivalent of a specified hexadecimal number.

New language elements (continued)

Element	Description
INSPECT()	Displays a dialog box that displays all the properties of a specified object and the current values of the properties.
ISTABLE()	Tests for the existence of a table in a specified database.
ITOH()	Returns the hexadecimal equivalent of a specified decimal number as a character string.
LDRIVER()	Returns the name of the language driver the current table or a specified table is using.
LENNUM()	Returns the display length of a specified number, including leading blanks.
LISTCOUNT()	Returns the number of prompts in a list box.
LISTSELECTED()	Returns the prompt of a list box item or combobox item selected by the user.
LOAD DLL	Initiates a DLL file, a precompiled library of external routines written in non-dBASE languages like C and Pascal.
LOCAL	Declares memory variables that are visible only in the subroutine where they're declared.
MD	Creates a new DOS directory.
MKDIR	Creates a new DOS directory.
MODIFY FORM	Opens the Form Designer, which creates or modifies a form file.
MODIFY MENU	Opens the Menu designer to modify a menu (.MNU) file.
NEXTKEY()	Returns the decimal value of a key or key combination held in the keyboard typeahead buffer.
OEM()	Returns a character string that is the OEM (code page) value of a specified ANSI character expression.
ON NETERROR	Executes a specified command when a multiuser-specific error occurs.
ON SELECTION FORM	Executes a subroutine or a codeblock when the user submits a form.
OPEN DATABASE	Establishes a connection to a database server.
OPEN FORM	Displays and enables forms and the objects they contain.
PLAY SOUND	Restores a sound from a .WAV file or from a memo field and plays it.
PROPER()	Converts a character string to proper-noun format.
PUTFILE()	Displays a dialog box in which you can create a new file name.
RANDOM()	Returns a random decimal value between 0 and 1.
READMODAL()	Opens a form as a modal window, and returns the name of the object that has input focus when the user submits the form.
REDEFINE	Modifies the settings made originally by the DEFINE command.
REFRESH	Updates the current or specified work area data buffers to reflect the latest changes to data.
RELATION()	Returns the key expression defined with the SET RELATION command for the current or specified work area.
RELEASE AUTOMEM	Clears all stored automem variables from memory.
RELEASE DLL	Deactivates DLL files, precompiled libraries of external routines that were previously initialized with LOAD DLL.
RELEASE OBJECT	Removes object definitions from memory.

New language elements (continued)

Element	Description
RENAME TABLE	Changes the name of a specified table.
REPLACE AUTOMEM	Transfers contents of memory variables into corresponding fields of the current record in the current table.
REPLACE BINARY	Replaces the contents of a binary field with the contents of a binary file.
REPLACE MEMO	Replaces the text of a memo field with the contents of an array.
REPLACE MEMO...FROM	Inserts a text file in a memo field.
REPLACE OLE	Inserts an OLE document into an OLE field.
RESOURCE()	Returns a Windows resource number found in a DLL file.
RESTORE IMAGE	Displays an image stored in a file or in a memo field.
ROLLBACK()	Replaces the ROLLBACK command.
SECONDS()	Returns the number of seconds that have elapsed since 12 AM (midnight).
SET COVERAGE	Determines whether dBASE creates or updates a .COV coverage file.
SET CUAENTER	Determines whether the Enter key works in Windows mode or dBASE DOS mode.
SET DATABASE	Sets the default database where tables are accessed from.
SET DATE TO	Sets the system date.
SET DBTYPE	Sets the default table type to either Paradox or dBASE.
SET DEVICE	? and <filename skeleton> options added.
SET EDITOR	Specifies the text editor to use when creating and editing programs and text files.
SET ERROR	Specifies one character expression to precede error messages and another character expression to follow them.
SET HELP	Determines which help file (.HLP) the dBASE help system uses.
SET KEY	Assigns a program or procedure to a specified key or key combination.
SET LDCONVERT	Determines whether data read from and written to character and memo fields is transliterated when the table character set does not match the global language driver.
SET PCOL	Sets the printing column position of a printer, which is the value of PCOL().
SET PROW	Sets the current row position of a printer's print head, which is the value of PROW().
SET TIME	Sets the system time.
SET TOPIC	Determines the help information that initially displays in the help window when you issue the HELP command or press F1.
SET WP	Specifies the text editor to use for viewing and editing memo fields.
SETTO()	Returns the current setting of a SET...TO command or function key.
SHELL()	Removes or restores the dBASE interactive environment.
SHOW OBJECT	Refreshes an object on the screen.
SLEEP	Pauses a program for a specified interval or until a specified time.
SQLERROR()	Returns the number of the last server error.

New language elements (continued)

Element	Description
SQLMESSAGE()	Returns the most recent server error message.
SQLEXEC()	Executes an SQL statement in the current database.
STATIC	Declares memory variables active only in the subroutine where they're declared but whose values remain in memory until the end of a dBASE session.
STORE AUTOMEM	Stores the contents of all the current record's fields to a set of memory variables.
STORE MEMO	Stores the text of a memo field to an array-type memory variable.
TARGET()	Returns the name of a table linked to the current or specified work area.
UPDATED()	Indicates whether you changed the contents of any @...GET fields or memory variables in the Command window.
VALIDDRIVE()	Indicates whether a specified drive exists and can be read.

dBASE also provides standard classes that let you create common windows controls such as pushbuttons, radio buttons, and entry fields. (See Chapter 7, "Classes" for a description of each class.)

Enhanced language elements

Enhanced language elements

Element	Change
ACTIVATE MENU	RETRACTED option added; causes a pop-up menu associated with a pad to be displayed only when the user presses Enter or clicks on a highlighted pad. Without RETRACTED, the pop-up menu is displayed as soon as the associated pad is highlighted.
ACTIVATE SCREEN	SAVE option added; prevents dBASE IV-style windows in the result pane of the Command window from scrolling up and out of the results pane of the Command window.
APPEND	NOWAIT and COLUMNAR options added.
APPEND FROM	<filename skeleton> and POSITION options added. PARADOX and DBASE types added. dBASE II, RPD, FW2, SYLK, DIF, and WKS options dropped.
APPEND MEMO	? and <filename skeleton> options added.
AT()	Memo field search no longer limited to the first 64K of data.
BROWSE	<browse name> option added. FOR and WHILE options added. COLOR option added. KEY...[EXCLUDE] option added. NOFOLLOW, NOINIT, NOMODIFY, NORMAL, NOTOGGLE, and NOWAIT options added. TITLE option added.
CHANGE	Same as EDIT.

Enhanced language elements (continued)

Element	Change
CLEAR	CHARACTER <i><expC></i> option added; fills the results pane of the Command window, or the current dBASE IV-style window, with the first character of the expression <i><expC></i> .
CLOSE DATABASES	Besides closing open tables, also closes open databases.
COPY	PARADOX and DBASE types added. dBASE II, RPD, FW2, SYLK, DIF, and WKS options dropped.
COPY FILE	? and <i><filename skeleton></i>
COPY INDEXES	? option added.
COPY MEMO	? option added.
COPY STRUCTURE	[TYPE] PARADOX DBASE option added.
COPY TO...STRUCTURE EXTENDED	[TYPE] PARADOX DBASE option added.
CREATE	? option added. [TYPE] PARADOX DBASE option added.
CREATE...FROM	? and <i><filename skeleton></i> options added. [TYPE] PARADOX DBASE option added.
CREATE LABEL	<i><filename skeleton></i> option added.
CREATE QUERY	Invokes the Query Designer. Update (.UPD) queries no longer supported.
CREATE REPORT	<i><filename skeleton></i> option added.
CREATE VIEW	Invokes the Query Designer. Update (.UPD) queries no longer supported.
DEFINE BAR	SKIP [FOR <i><condition expL></i>] option added; makes the pop-up menu item conditionally unavailable for selection.
DEFINE PAD	SKIP [FOR <i><condition expL></i>] option added; makes the menu pad conditionally unavailable for selection.
DEFINE POPUP	PROMPT ARRAY added; PROMPT ARRAY <i><array name></i> causes the pop-up menu to display the value of each element in the specified array.
DELETE FILE	<i><filename skeleton></i> option added.
DIFFERENCE()	<i><memo field></i> option added.
DISKSPACE()	<i><drive expN></i> option added; specifies a drive number from 1 to 26. For example, the numbers 1 and 2 correspond to drives A and B, respectively.
DISPLAY FILES	ON <i><drive></i> and TO FILE ? options added.
DISPLAY MEMORY	? and <i><filename skeleton></i> options added.
DISPLAY STRUCTURE	? and IN <i><alias></i> options added.
DISPLAY STATUS	? and <i><filename skeleton></i> options added.
DO	calling a UDF with DO is supported.
EDIT	FOR and WHILE options added. <i><bookmark></i> option added. COLOR option added. COLUMNAR, NOFOLLOW, NOINIT, NOMODIFY, NORMAL, NOTOGGLE, and NOWAIT options added. TITLE option added.
ERASE	<i><filename skeleton></i> option added.
GO/GOTO	<i><bookmark></i> option added.

Enhanced language elements (continued)

Element	Change
IF	ELSEIF option added (ELSE IF also allowed).
IMPORT	WB1 option added. dBASE II, RPD, FW2, FW3, PFS, and WKS options dropped.
INDEX	? and <filename skeleton> options added. PRIMARY key added to allow creating primary indexes on Paradox tables.
INKEY()	<mouse expC> option added.
ISALPHA()	<memo field> option added.
ISLOWER()	<memo field> option added
ISUPPER()	<memo field> option added
JOIN	? option added. [TYPE] PARADOX DBASE option added.
LABEL FORM	<filename skeleton> option added.
LEFT()	Return value no longer limited to 254 characters; memo field option added.
LEN()	Null characters are counted.
LIKE()	<memo field> option added.
LIST STRUCTURE	? and IN <alias> options added.
LOCK()	<bookmark> option added.
LOWER()	<memo field> option added.
LTRIM()	<memo field> option added. Return value no longer limited to 254 characters.
MAX()	Logical expressions supported.
MEMLINES()	<line length exp> option added.
MIN()	Logical expressions supported.
MLINE()	<line length exp> option added.
MODIFY COMMAND	? and <filename skeleton> options added.
MODIFY QUERY	Invokes the Query Designer. Update (.UPD) queries no longer supported.
MODIFY VIEW	Invokes the Query Designer. Update (.UPD) queries no longer supported.
OS()	Optional <expN> argument returns the version of Windows currently running.
PRIVATE	LIKE and EXCEPT supported in the same statement.
QUIT	Allowable <expN> value now 1 to 254 instead of 0 to 255.
RELEASE	LIKE and EXCEPT supported in the same statement.
REPLICATE()	<memo field> option added. Return value no longer limited to 254 characters.
REPORT FORM	<filename skeleton> option added.
RESTORE	? and <filename skeleton> options added.
RESTORE WINDOW	? and <filename skeleton> options added
RIGHT()	<memo field> option added. Return value no longer limited to 254 characters.
RLOCK()	<bookmark> option added

Enhanced language elements (continued)

Element	Change
RTRIM()	<memo field> option added. Return value no longer limited to 254 characters.
SAVE	? and <filename skeleton> option added. Both LIKE and EXCEPT supported in the same SAVE statement.
SAVE SCREEN	TO FILE option added, and TO <memovar> is no longer a required option.
SCAN	Nested SCAN loops
SEEK	<exp list> to allow searching for composite field keys.
SET	Automatically writes changes to DBASEWIN.INI.
SET ALTERNATE	? and <filename skeleton> options added
SET CURRENCY	<exp C> option added; sets the characters that display as a currency sign.
SET DEVICE	? and <filename skeleton> options added
SET FILTER	<filename skeleton> option added.
SET INDEX	<filename skeleton> option added. TAG keyword option added.
SET KEY TO	LOW, HIGH, and EXCLUDE keyword options added.
SET LIBRARY	? and <filename skeleton> options added.
SET MESSAGE	The AT option is not supported.
SET PRECISION	dBASE now provides 19 digits of precision for both numeric and float data types.
SET PROCEDURE	?, <filename skeleton>, and ADDITIVE options supported; ADDITIVE opens the procedure file(s) without closing any you've opened with previous SET PROCEDURE statements.
SET RELATION	<exp list> option added to allow setting relation on composite field keys. CONSTRAIN and INTEGRITY[CASCADE] options added to ensure data integrity in linked tables.
SET VIEW	Activates a query created with Query Designer.
SPACE()	Return value no longer limited to 254 characters.
STUFF()	<memo field> option added.
SORT	? option added. [TYPE] PARADOX DBASE option added.
SOUNDEX()	<memo field> option added.
STR()	<exp C> option added; sets the character to pad the beginning of the returned character string with.
STUFF()	<memo field> option added.
SUBSTR()	<memo field> option added. Return value no longer limited to 254 characters. Different behavior if <start expN> or <length expN> is zero or if <length expN> is a negative number.
TRIM()	<memo field> option added. Return value no longer limited to 254 characters.
TYPE	MORE, ?, and <filename skeleton> options added

Enhanced language elements (continued)

Element	Change
UPPER()	<memo field> option added
USE	<filename skeleton> and SHARED options added. [TYPE] PARADOX DBASE option added. NOLOG option dropped.

Unsupported language elements

Unsupported language elements

Element	If used, dBASE...
_pecode	Returns a null string
_pscode	Returns a null string
_pwait	Returns .F.
ASSIST	Returns a warning
BEGIN...END TRANSACTION	Returns a warning
CALL	Returns a warning
CALL()	Returns a warning
COMPLETED()	Returns .T.
DEXPORT	Returns a warning
DGEN()	Returns 0
DISPLAY HISTORY	Ignores this command
DISPLAY USERS	Ignores this command
EXPORT	Returns a warning
FIXED()	Returns the value that was passed to it
ISMARKED()	Returns .F.
LIST HISTORY	Ignores this command
LIST USERS	Ignores this command
LOAD	Returns a warning
LOGOUT	Ignores this command
PLAY MACRO	Returns a warning
RELEASE MODULE	Returns a warning
RESET	Returns a warning
RESTORE MACROS	Returns a warning
ROLLBACK	Returns a warning
SAVE MACROS	Returns a warning
SET CLOCK	Returns a warning
SET COLOR ON OFF	Returns a warning
SET DBTRAP	Returns a warning
SET DEBUG	Returns a warning
SET HELP ON OFF	Returns a warning
SET HISTORY	Returns a warning
SET HOURS	Returns a warning
SET INSTRUCT	Returns a warning
SET PAUSE	Returns a warning
SET SCOREBOARD	Returns a warning
SET SQL	Returns a warning
SET STATUS	Returns a warning
SET TRAP	Returns a warning

B

Visual dBASE specifications

This appendix outlines the specifications for *Visual* dBASE files and operations.

.DBF tables

Table B.1

Item description	dBASE IV limit	Visual dBASE limit
Maximum number of records	1 billion	1 billion
Maximum size of .DBF table file (in bytes)	2 billion	2 billion
Maximum record size (in bytes)	4000 (not including _DBASELOCK field)	32,767 (including _DBASELOCK field)
Maximum number of fields in .DBF table file	255	1024

Indexes

Table B.2

Item description	dBASE IV limit	Visual dBASE limit
Maximum index key expression length (in bytes)	220	220
Maximum index FOR expression length (in bytes)	220	220
Maximum evaluated index key length (in bytes)	100	100
Maximum blocksize in .MDX index file (in bytes)	16,384	32,256 (default 1024)
Maximum number of index tags in .MDX index file	47	47
Fixed blocksize of .NDX index file (in bytes)	512	512

Fields

Table B.3

Item description	dBASE IV limit	Visual dBASE limit
Maximum size of character fields (in bytes)	254	254
Size of date fields (in bytes)	8	8
Size of logical fields (in bytes)	1	1
Maximum size of numeric fields (in digits)	20	20
Maximum size of memo fields (in bytes); includes OLE and binary type fields	Memo fields limited only by available memory	Limited only by available memory
Maximum characters in field names	10	10
Numeric accuracy of type F numbers (digits)	15.9	19
Largest type F number	0.9xE+308	0.9xE+308
Smallest type F number	0.1xE-307	0.1xE-307
Numeric accuracy of type N numbers (digits)	20	19
Largest type N number	0.9xE+308	0.9xE+308
Smallest type N number	0.1xE-307	0.1xE-307

Multiusers

Table B.4

Item description	dBASE IV limit	Visual dBASE limit
Maximum number of locks (files and records)	200 total	100 per work area
Maximum number of reprocess counts	32,000	32,000
Maximum number of seconds for refresh	3600	3600

Procedures

Table B.5

Item description	dBASE IV limit	Visual dBASE limit
Maximum length of command line in programs (in bytes)	1024	4096
Maximum number of active .PRO files	32	147
Maximum size of a procedure (in bytes)	65,520	Limited only by available memory
Maximum number of procedures per file	963	193 ¹
Maximum number of open procedure files	1	Limited only by available memory

1. Because you can open multiple procedure files with the ADDITIVE option, the maximum number of procedures available is limited only by available memory.

Files

Table B.6

Item description	dBASE IV limit	Visual dBASE limit
Maximum number of open files (of all types)	99	250
Maximum number of open tables	40	225
Maximum number of open .DBT memo files per active .DBF table	1	1
Maximum number of open .MDX index files per active .DBF table	1 production .MDX plus additional non-production indexes allowed.	1 production .MDX plus additional non-production indexes allowed
Maximum number of open .NDX index file per active .DBF table	10	10
Maximum number of open .FMT format files per active .DBF table	1	1
Maximum width of a report	255 characters	22.75 inches
Maximum number of tables per report	9	225
Maximum number of reports per table	Unlimited	Unlimited
Maximum number of pages in a report	32,767	Unlimited
Maximum number of nested report group bands	44	Unlimited
Maximum number of fields in a report	Limited only by available memory	Unlimited

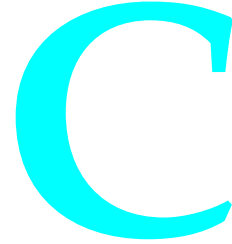
Miscellaneous

Table B.7

Item description	dBASE IV limit	Visual dBASE limit
Maximum number of lines in Text Editor	32,000	Unlimited
Maximum line length in Text Editor	1024	32,767
Maximum width of a form	80 characters	32767 pixels (the number of characters depends on the pixel width of the font)
Maximum number of rows in a form	32,767	32,767 pixels
Maximum command line length entered in the Command window	255	4096
Maximum number of lines stored in input pane of Command window	N/A	1000 lines
Maximum number of simultaneous sort levels	16	16
Maximum number of printer drivers	4	Unlimited
Maximum number of fonts per printer driver	5	9 (total maximum; not per driver)
Maximum number of work areas	40	225
Number of programmable function keys	29	29
Maximum memory variable size	32,767	32,767
Maximum number of array dimensions	2	255

Table B.7 (continued)

Item description	dBASE IV limit	<i>Visual</i> dBASE limit
Maximum elements per dimension	65,525	Unlimited
Maximum total size of array	Depends on available memory	Limited only by available memory



File structures

This appendix provides information on the structure of the dBASE table (.DBF) and the memo (.DBT) files.

Table header and records

A dBASE table file (.DBF) 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.

Table header structure

The header structure, detailed in Table C.1 and Table C.2, provides information that *Visual* dBASE uses to maintain the table file.

Table C.1 dBASE table file header

Byte	Contents	Description
0	1 byte	Valid <i>Visual</i> dBASE table file; bits 0–2 indicate version number; bit 3 indicates presence of a dBASE IV or <i>Visual</i> dBASE memo file; bits 4–6 indicate the presence of a dBASE IV SQL table; bit 7 indicates the presence of any .DBT memo file (either a dBASE III PLUS type or a dBASE IV or <i>Visual</i> dBASE memo file)
1–3	3 bytes	Date of last update; in YYMMDD format
4–7	32-bit number	Number of records in the table
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; filled with zeros
14	1 byte	Flag indicating incomplete dBASE IV transaction ¹
15	1 byte	dBASE IV encryption flag ²
16–27	12 bytes	Reserved for multi-user processing

Table C.1 dBASE table file header (continued)

Byte	Contents	Description
28	1 byte	Production .MDX flag; 01H stored in this byte if a production .MDX file exists for this table; 00H stored if no .MDX file exists
29	1 byte	Language driver ID
30–31	2 bytes	Reserved; filled with zeros
32 – n^3	32 bytes each	Field descriptor array (the structure of this array is shown in Table C-2).
$n + 1$	1 byte	0DH stored as the field terminator

1. Flag not used by *Visual* dBASE; in dBASE IV, BEGIN TRANSACTION sets this flag to 01H, END TRANSACTION and ROLLBACK resets it to 00H.
2. Encryption not supported in *Visual* dBASE; in dBASE IV, if flag is set to 01H, the table is encrypted.
3. n is the last byte in the field descriptor array. The size of the array depends on the number of fields in the table file.

Table C.2 Table field descriptor bytes

Byte	Contents	Description
0–10	11 bytes	Field name in ASCII (zero-filled)
11	1 byte	Field type in ASCII (B, C, D, F, G, 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 field is not indexed

Table records

The records follow the header in the table 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 OEM Code Page character value of 26 (1AH). You can input OEM code page data as indicated in Table C.3.

Table C.3 Allowable input for dBASE data types

Data type	Data input
B (Binary)	All OEM code page characters (stored internally as 10 digits representing a .DBT block number)
C (Character)	All OEM code page characters
D (Date)	Numbers and a character to separate month, day, and year (stored internally as 8 digits in YYYYMMDD format)

Table C.3 Allowable input for dBASE data types (continued)

Data type	Data input
G (General or OLE)	All OEM code page characters (stored internally as 10 digits representing a .DBT block number)
N (Numeric) and F (Floating Point)	– . 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 OEM code page characters (stored internally as 10 digits representing a .DBT block number)

Binary, memo, and OLE fields and .DBT files

Binary, memo, and OLE fields store data in .DBT files consisting of blocks numbered sequentially (0, 1, 2, and so on). SET BLOCKSIZE determines the size of each block. The first block in the .DBT file, block 0, is the .DBT file header.

Each binary, memo, or OLE field of each record in the .DBF file contains the number of the block (in OEM code page values) where the field's data actually begins. If a field contains no data, the .DBF file contains blanks (20H) rather than a number.

When data is changed in a 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 (or binary and OLE fields), *Visual* dBASE (like dBASE IV) may reuse the space from the deleted text when you input new text. dBASE III PLUS always appends new text to the end of the .DBT file. In dBASE III PLUS, the .DBT file size grows whenever new text is added, even if other text in the file is deleted.

D

INKEY() and READKEY() values

This appendix provides information on the values returned by the INKEY() and READKEY() functions.

Table D.1 INKEY() return values

Key pressed	Return value	Shift+key return value	Ctrl+Key return value	Alt+key ¹ return value
0	48	Depends on keyboard	-404	-452
1	49	Depends on keyboard	-404	-451
2	50	Depends on keyboard	-404	-450
3	51	Depends on keyboard	-404	-449
4	52	Depends on keyboard	-404	-448
5	53	Depends on keyboard	0	-447
6	54	Depends on keyboard	-30	-446
7	55	Depends on keyboard	-404	-445
8	56	Depends on keyboard	-404	-444
9	57	Depends on keyboard	-404	-443
a	97	65	1	-435
b	98	66	2	-434
c	99	67	3	-433
d	100	68	4	-432
e	101	69	5	-431

Table D.1 INKEY() return values (continued)

Key pressed	Return value	Shift+key return value	Ctrl+Key return value	Alt+key ¹ return value
f	102	70	6	-430
g	103	71	7	-429
h	104	72	8	-428
i	105	73	9	-427
j	106	74	10	-426
k	107	75	11	-425
l	108	76	12	-424
m	109	77	13	-423
n	110	78	14	-422
o	111	79	15	-421
p	112	80	16	-420
q	113	81	17	-419
r	114	82	18	-418
s	115	83	19	-417
t	116	84	20	-416
u	117	85	21	-415
v	118	86	22	-414
w	119	87	23	-413
x	120	88	24	-412
y	121	89	25	-411
z	122	90	26	-410
F1 (Ctrl+ \backslash)	28	-20	-10	-30
F2	-1	-21	-11	-31
F3	-2	-22	-12	-32
F4	-3	-23	-13	-33
F5	-4	-24	-14	-34
F6	-5	-25	-15	-35
F7	-6	-26	-16	-36
F8	-7	-27	-17	-37
F9	-8	-28	-18	-38
F10	-9	-29	-19	-39
F11	-544	-546	-548	-550
F12	-545	-547	-549	-551
Left Arrow	19	-500	1	0
Right Arrow	4	-501	6	0
Up Arrow	5	5	5	0
Down Arrow	24	24	24	0
Home (Ctrl+)]	26	26	29	0
End	2	2	23	0
Tab	9	-400	0	0
Enter	13	0	-402	0

Table D.1 INKEY() return values (continued)

Key pressed	Return value	Shift+key return value	Ctrl+Key return value	Alt+key ¹ return value
Esc (Ctrl+I)	27	27	-	-
Ins	22	0	0	0
Del	7	-502	7	7
Backspace	127	127	-401	-403
PgUp	18	18	31	0
PgDn	3	3	30	0

1. The Alt+key value returned for all character keys, except lower-case letters a through z, is the character value minus 500. For lower-case letters, the Alt+key values are the same as those for upper-case letters.

Table D.2 READKEY() values

Non-updated code number	Updated code number	Key pressed	Description
0	256	Ctrl+S, Left-arrow, Ctrl+H	Move back one character
—	256	Backspace	Move back one character
1	257	Ctrl+D, Right-arrow, Ctrl+L	Move forward one character
2	258	Ctrl+A, Ctrl+Left arrow	Move to beginning of previous word
3	259	Ctrl+F, Ctrl+Right arrow	Move to beginning of next word
4	260	Ctrl+E, Ctrl+K, Up-arrow	Move back one field
5	261	Ctrl+J, Ctrl+X, Down-arrow	Move forward one field
6	262	Ctrl+R, PgUp	Move back one screen
7	263	Ctrl+C, PgDn	Move forward one screen
12	—	Ctrl+Q, Esc	Terminate without saving
—	270	Ctrl+W, Ctrl+End	Terminate with save
15	271	Enter, Ctrl+M	Return or fill last record
16	—	Enter, Ctrl+M	Move to beginning of record in an APPEND window
33	289	Ctrl+Home	Menu display toggle
34	290	Ctrl+PgUp	Zoom out
35	291	Ctrl+PgDn	Zoom in
36	292	F1	Help function key

E

ASCII character chart (code page 437)

Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char
0	00	<null>	32	20	<space>	64	40	@	96	60	`
1	01	A	33	21	!	65	41	A	97	61	a
2	02	B	34	22	"	66	42	B	98	62	b
3	03	C	35	23	#	67	43	C	99	63	c
4	04	D	36	24	\$	68	44	D	100	64	d
5	05	E	37	25	%	69	45	E	101	65	e
6	06	F	38	26	&	70	46	F	102	66	f
7	07	G	39	27	'	71	47	G	103	67	g
8	08	H	40	28	(72	48	H	104	68	h
9	09	I	41	29)	73	49	I	105	69	i
10	0A	J	42	2A	*	74	4A	J	106	6A	j
11	0B	K	43	2B	+	75	4B	K	107	6B	k
12	0C	L	44	2C	,	76	4C	L	108	6C	l
13	0D	M	45	2D	-	77	4D	M	109	6D	m
14	0E	N	46	2E	.	78	4E	N	110	6E	n
15	0F	O	47	2F	/	79	4F	O	111	6F	o
16	10	P	48	30	0	80	50	P	112	70	p
17	11	Q	49	31	1	81	51	Q	113	71	q
18	12	R	50	32	2	82	52	R	114	72	r
19	13	S	51	33	3	83	53	S	115	73	s
20	14	T	52	34	4	84	54	T	116	74	t
21	15	U	53	35	5	85	55	U	117	75	u
22	16	V	54	36	6	86	56	V	118	76	v
23	17	W	55	37	7	87	57	W	119	77	w
24	18	X	56	38	8	88	58	X	120	78	x
25	19	Y	57	39	9	89	59	Y	121	79	y
26	1A	Z	58	3A	:	90	5A	Z	122	7A	z
27	1B	[59	3B	;	91	5B	[123	7B	{
28	1C	\	60	3C	<	92	5C	\	124	7C	
29	1D]	61	3D	=	93	5D]	125	7D	}
30	1E	^	62	3E	>	94	5E	^	126	7E	~
31	1F	_	63	3F	?	95	5F	_	127	7F	△

Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char
128	80	Ç	160	A0	á	192	C0	À	224	E0	à
129	81	ü	161	A1	í	193	C1	Á	225	E1	á
130	82	é	162	A2	ó	194	C2	Â	226	E2	â
131	83	â	163	A3	ú	195	C3	Ã	227	E3	ã
132	84	ä	164	A4	ñ	196	C4	Ä	228	E4	ä
133	85	à	165	A5	Ñ	197	C5	Å	229	E5	å
134	86	â	166	A6	ı	198	C6	Æ	230	E6	æ
135	87	ç	167	A7	§	199	C7	Ç	231	E7	ç
136	88	ê	168	A8	ç	200	C8	È	232	E8	è
137	89	ë	169	A9	©	201	C9	É	233	E9	é
138	8A	è	170	AA	ª	202	CA	Ê	234	EA	ê
139	8B	ï	171	AB	«	203	CB	Ë	235	EB	ë
140	8C	î	172	AC	¬	204	CC	Ì	236	EC	∞
141	8D	ì	173	AD	ı	205	CD	Í	237	ED	í
142	8E	Ä	174	AE	®	206	CE	Î	238	EE	î
143	8F	Å	175	AF	-	207	CF	Ï	239	EF	ï
144	90	É	176	B0	º	208	D0	Ð	240	F0	≡
145	91	æ	177	B1	±	209	D1	Ñ	241	F1	±
146	92	Æ	178	B2	²	210	D2	Ò	242	F2	≥
147	93	ô	179	B3	³	211	D3	Ó	243	F3	≤
148	94	ö	180	B4	´	212	D4	Ô	244	F4	∫
149	95	ò	181	B5	µ	213	D5	Õ	245	F5	∫
150	96	û	182	B6	¶	214	D6	Ö	246	F6	÷
151	97	ù	183	B7	·	215	D7	×	247	F7	≈
152	98	ÿ	184	B8	¸	216	D8	Ø	248	F8	º
153	99	Ö	185	B9	¹	217	D9	Ù	249	F9	•
154	9A	Ü	186	BA	º	218	DA	Ú	250	FA	•
155	9B	›	187	BB	»	219	DB	Û	251	FB	√
156	9C	£	188	BC	¼	220	DC	Ü	252	FC	n
157	9D	¥	189	BD	½	221	DD	Ý	253	FD	²
158	9E	℞	190	BE	¾	222	DE	Þ	254	FE	Ý
159	9F	f	191	BF	¿	223	DF	ß	255	FF	

F

Error codes

Table F.1 Error codes and associated messages

Code	Message
1	Error creating file
2	Error opening file
3	Error closing file
4	End of table encountered
5	Record out of range
6	Error positioning in file
7	File does not exist
8	File already exists
9	File already open
10	Unable to rename file
11	Structure invalid
12	Invalid COV file
13	dBASE IV binary report file not supported - use component builder to convert it
14	Invalid label file
15	dBASE IV binary label file not supported - use component builder to convert it
16	Invalid memory variable file
17	Invalid PRO / FMO file
18	Invalid query file
19	Invalid report file
20	Invalid driver name or insufficient system resources
21	Invalid view file
22	Invalid window file
23	Operation not allowed for calculated fields
24	Operation not allowed on read-only files
25	Bad field name
26	Bad field type

Table F.1 Error codes and associated messages (continued)

Code	Message
28	Duplicate field name
29	Error writing file
30	Not a valid dBASE table
31	No such record in index
32	Illegal key
33	WindowMenu must be on child of MenuBar
34	No table in use in area
35	Table is not indexed
36	Maximum number of fields reached
40	Field not found
41	Cyclic RELATION not allowed
42	Work area already used in relation
44	Too many RELATIONS in this chain
47	Too many index files
48	Invalid order number
49	No fields were found to process
50	Field must be a memo field
51	Field must be a binary field
53	Tag not found
54	Unrecognized command verb
55	Command too large
56	Expression expected
57	Expression too large
58	Too few arguments. Expecting at least
59	Too many arguments. Expecting at most
61	Unterminated string
62	Unbalanced parentheses
63	Syntax error
67	Something is missing. Expecting
68	Unknown keyword
70	PARAMETERS command must be at top of function or procedure
71	Invalid name character
74	ENDIF encountered without preceding IF
75	Missing ENDIF
76	ENDDO encountered without preceding WHILE
77	Missing ENDDO
78	NEXT encountered without preceding FOR
79	Missing NEXT
80	ENDSCAN encountered without preceding SCAN
81	Missing ENDSCAN
82	UNTIL encountered without preceding DO
83	Missing UNTIL

Table F.1 Error codes and associated messages (continued)

Code	Message
84	ENDCASE encountered without preceding CASE
85	Missing ENDCASE
86	ENDPRINTJOB command encountered without previous PRINTJOB command
87	Missing ENDPRINTJOB
88	ENDCLASS/PROTECT command encountered without previous CLASS command
89	Missing ENDCLASS
90	ENDTEXT command encountered without previous TEXT command
91	Missing ENDTEXT
94	Loop stack overflow.
95	Too many nested FOR loops.
96	Too many nested SCAN loops.
97	Unallowed phrase/keyword in command
98	Keyword Repeated
100	UDF must return a value
101	Too many dimensions
102	Too many UDF/PROCEDURES defined in program
103	Invalid FUNCTION or PROCEDURE name
104	Invalid CLASS name
105	Invalid MEMBER name
106	Program too big to compile
107	Not enough memory for this operation
108	In use by another
109	Record is in use by another
110	Command requires exclusive use of table
111	Memory variable space exhausted
112	Not enough memory for DOS
114	Filename space exhausted
115	Only valid in program files
116	No PARAMETERS statement found
117	Unmatched parameters
118	Program not SUSPENDED
119	No such bar
120	No such menu
121	No such pad
122	No such popup
124	No such window
125	No such window
126	No such form object
127	Menu already active
128	Popup already active
129	Unable to add data while constraints active
130	Windows print file name longer than 31 characters

Table F.1 Error codes and associated messages (continued)

Code	Message
131	Printer is either not connected or turned off
132	Window out of range
133	Unauthorized access level
134	No bars defined for popup/pulldown
135	Bars already defined for popup.
136	Bars must have a positive value.
139	Cannot release active
140	Datasource/Prompt cannot be MEMO/OLE/BINARY
142	Cannot change property while form is open
143	Expecting reference to MENU object
144	First class in menu file is not derived from MENU
145	Cannot have more than one form object with the same name
146	Internal stack overflow
147	Internal stack underflow
148	Stack overflow
149	Runtime buffer overflow
150	String buffer overflow
151	Attempt to free a bad memory block
152	Attempt to load a bad icode block
153	Macros cannot expand flow of control commands
154	Expanded macro variable does not return a valid identifier
155	Cannot assign to reserved word NULL
156	Numbers are not allowed in the CURRENCY symbol
157	Illegal work area number or alias
158	Illegal value
159	UDF or procedure already exists
160	Unable to execute DOS
161	Too many nested expressions
162	Nested views not allowed
163	Data type mismatch. Expecting
164	Out of range
167	Variable undefined
168	Not an array
169	Illegal Opcode
172	Maximum number of nested FOR NEXT loops exceeded
175	Maximum number of DO or UDF parameters exceeded
178	Alias not found
179	MEMO field not allowed here
180	ALIAS already in use
181	Processing would exceed maximum allowed string length
182	Procedure not found
185	Illegal file name

Table F.1 Error codes and associated messages (continued)

Code	Message
186	Beginning of table encountered
187	Error reading file
188	Unexpected type
189	Printer error
190	Memory variable already defined - cannot make PUBLIC
191	CONTINUE without previous LOCATE
192	Value out of range
193	Invalid subscript reference
196	Invalid printer redirection
197	Cannot execute this command when DESIGN is off
198	Command not functional in this release of dBASE for Windows
199	Restricted command: not allowed in this context
200	Command will never be reached
201	Command not functional in dBASE for Windows
202	Extra characters ignored at end of command
203	Program was previously compiled with SET COVERAGE OFF
206	Drive not ready
207	UDF or PROCEDURE not found
209	Too many files open
210	Invalid directory
211	Invalid disk drive
212	Cannot redefine active menu
214	No such listbox
215	Window not active
217	Symbol table space exhausted. Increase to
218	SET FIELDS space exhausted
219	No previous DO WHILE/SCAN/PRINTJOB/FOR to match this command
221	Too many nested DO/UDF
222	Maximum number of locked records exceeded
223	Sharing buffers are full
224	Error unlocking file
225	Unmatched #else
226	Unmatched #endif
227	Maximum #ifdef nesting exceeded
228	Expecting #endif
229	Preprocessor expansion too large
230	Include file not found
231	Table already open
232	Database already open
234	Operation not allowed in transaction
236	Operation not allowed on this table
237	Index is not open

Table F.1 Error codes and associated messages (continued)

Code	Message
239	IDAPI Error
240	Server Error
241	Database not opened
242	Invalid value for convert size (8-24)
243	Invalid file Handle
244	IDAPI Not Initialized
245	Cannot UPDATE a table with itself
246	Invalid Catalog
247	Invalid password
248	Access denied
249	Can only change draft mode on page boundaries
250	Can only change page orientation on page boundaries
251	Already in printjob
252	Wrong version of IDAPI01.DLL
253	No print driver selected
254	No pads defined for
255	AUTOEXTERN not supported for this database
256	Output parameter required
257	Attempting to call a method as a function
258	Method is not available on object
259	No Records Selected
260	Internal Exception Error
261	Stored procedures not supported
262	Form cannot be MDI
265	Resource not found
266	Cannot load print driver
267	Cannot paste more than one file
268	Cannot recognize dropped file
269	Cannot Paste selection
270	Cannot Copy selection
271	Cannot Package file
272	Cannot activate object. OLE server is busy
273	Cannot update linked object
274	Unknown error saving window contents
275	Cannot perform operation on static object
276	Error connecting to OLE server. May be bad object path if a link.
277	Invalid command verb for OLE object
278	OLE object error
279	OLE BLOB field is corrupted
280	OLE BLOB field data is from an incompatible version
281	Attempt to access released object
282	Property is read only

Table F.1 Error codes and associated messages (continued)

Code	Message
283	only 1 or 2 dimensional array is allowed in this operation
284	Report Engine Error
285	Property not found
286	Operation not allowed on read-only fields
287	Operation not allowed on read-only tables
288	An Editor or Viewer of a memo field is still open
289	Not member of Class or Base class
290	SUPER not allowed when THIS is undefined
291	Unable to open form
292	Unable to create control
293	No such form
294	The system registry does not contain an OLE server for a file with extension
295	OLE: cannot create link
298	Constant is already #defined
299	Field must be an OLE field
301	Position not in window
302	Invalid Color
303	Parameter type '...' can only be used with CDECL calling convention
304	DLL does not support Multiple Instances
305	Error loading DLL
306	Extern
307	Parent is not a REPORT
308	Parent is not a PAGETEMPLATE or BANDBODY
309	Error Saving VBX
310	BINARY field not allowed here
311	OLE field not allowed here
312	Popup too small
313	Error creating palette
314	OLE Error
315	only 1 dimensional array is allowed in this operation
316	Incomplete link specification
317	Selected tables cannot be related
318	Too many symbols in this module
319	Cannot create directory
320	Invalid table name
321	Invalid preprocessor identifier
322	DLL not Loaded
323	Invalid key label
324	Only allowed in function or procedure scope
325	Not a valid table
326	Port not configured for a printer
327	#includes nested too deeply

Table F.1 Error codes and associated messages (continued)

Code	Message
328	Index expressions not allowed for INTEGRITY rules
329	Related records still exist in alias
330	SET KEY active in alias
331	Relation using CONSTRAIN
332	No matching parent record
333	Operation not allowed across different databases or table types
334	Key already exists in parent
335	First class in .WFM file is not derived from FORM
336	Relation expression and active index expression must be the same
337	Memo file does not exist
338	Production index file does not exist
339	Invalid file privileges
340	Form already open
341	Error reading from binary field
342	Must convert report before modifying
343	Class does not exist
344	Fix or remove errors before running query
345	PRIMARY must start with first field
346	Fields must be in consecutive order
347	Report writer has not been installed
348	VBX dlls cannot be RELEASED
349	VBStream file Missing or Corrupt
350	Cannot JOIN table with itself
351	Cannot assign to reserved word THIS
352	OLE Unknown interface
353	OLE Member Not found
354	OLE Parameter Not found
355	OLE Data Type Mismatch
356	OLE Unknown name
357	OLE No Named arguments
358	OLE Bad Variable Type
359	OLE Dispatch Exception
360	OLE Overflow
361	OLE Invalid Subscript
362	OLE Unknown Class
363	OLE Array is locked
364	OLE Bad parameter count
365	OLE Parameter not optional
366	OLE Bad call
367	OLE Not a collection
368	OLE Unknown error
369	OLE Object does not support automation

Table F.1 Error codes and associated messages (continued)

Code	Message
370	OLE Unable to create object
371	OLE Class name not in registry
372	In use by another
373	Record is in use by another
374	Property is not accessible

Table F.2 Error codes in alphabetical order

Message	Code
#includes nested too deeply	327
Access denied	248
ALIAS already in use	180
Alias not found	178
Already in printjob	251
An Editor or Viewer of a memo field is still open	288
Attempt to access released object	281
Attempt to free a bad memory block	151
Attempt to load a bad icode block	152
Attempting to call a method as a function	257
AUTOEXTERN not supported for this database	255
Bad field name	25
Bad field type	26
Bars already defined for popup.	135
Bars must have a positive value.	136
Beginning of table encountered	186
BINARY field not allowed here	310
Can only change draft mode on page boundaries	249
Can only change page orientation on page boundaries	250
Cannot activate object. OLE server is busy	272
Cannot assign to reserved word NULL	155
Cannot assign to reserved word THIS	351
Cannot change property while form is open	142
Cannot Copy selection	270
Cannot create directory	319
Cannot execute this command when DESIGN is off	197
Cannot have more than one form object with the same name	145
Cannot JOIN table with itself	350
Cannot load print driver	266
Cannot Package file	271
Cannot paste more than one file	267
Cannot Paste selection	269
Cannot perform operation on static object	275

Table F.2 Error codes in alphabetical order (continued)

Message	Code
Cannot recognize dropped file	268
Cannot redefine active menu	212
Cannot release active	139
Cannot UPDATE a table with itself	245
Cannot update linked object	273
Class does not exist	343
Command not functional in dBASE for Windows	201
Command not functional in this release of dBASE for Windows	198
Command requires exclusive use of table	110
Command too large	55
Command will never be reached	200
Constant is already #defined	298
CONTINUE without previous LOCATE	191
Cyclic RELATION not allowed	41
Data type mismatch. Expecting	163
Database already open	232
Database not opened	241
Datasource/Prompt cannot be MEMO/OLE/BINARY	140
dBASE IV binary label file not supported - use component builder to convert it	15
dBASE IV binary report file not supported - use component builder to convert it	13
DLL does not support Multiple Instances	304
DLL not Loaded	322
Drive not ready	206
Duplicate field name	28
End of table encountered	4
ENDCASE encountered without preceding CASE	84
ENDCLASS/PROTECT command encountered without previous CLASS command	88
ENDDO encountered without preceding WHILE	76
ENDIF encountered without preceding IF	74
ENDPRINTJOB command encountered without previous PRINTJOB command	86
ENDSCAN encountered without preceding SCAN	80
ENDTEXT command encountered without previous TEXT command	90
Error closing file	3
Error connecting to OLE server. May be bad object path if a link.	276
Error creating file	1
Error creating palette	313
Error loading DLL	305
Error opening file	2
Error positioning in file	6
Error reading file	187
Error reading from binary field	341
Error Saving VBX	309

Table F.2 Error codes in alphabetical order (continued)

Message	Code
Error unlocking file	224
Error writing file	29
Expanded macro variable does not return a valid identifier	154
Expecting #endif	228
Expecting reference to MENU object	143
Expression expected	56
Expression too large	57
Extern	306
Extra characters ignored at end of command	202
Field must be a binary field	51
Field must be a memo field	50
Field must be an OLE field	299
Field not found	40
Fields must be in consecutive order	346
File already exists	8
File already open	9
File does not exist	7
Filename space exhausted	114
First class in .WFM file is not derived from FORM	335
First class in menu file is not derived from MENU	144
Fix or remove errors before running query	344
Form already open	340
Form cannot be MDI	262
IDAPI Error	239
IDAPI Not Initialized	244
Illegal file name	185
Illegal key	32
Illegal Opcode	169
Illegal value	158
Illegal work area number or alias	157
In use by another	108
In use by another	372
Include file not found	230
Incomplete link specification	316
Index expressions not allowed for INTEGRITY rules	328
Index is not open	237
Internal Exception Error	260
Internal stack overflow	146
Internal stack underflow	147
Invalid Catalog	246
Invalid CLASS name	104
Invalid Color	302

Table F.2 Error codes in alphabetical order (continued)

Message	Code
Invalid command verb for OLE object	277
Invalid COV file	12
Invalid directory	210
Invalid disk drive	211
Invalid driver name or insufficient system resources	20
Invalid file Handle	243
Invalid file privileges	339
Invalid FUNCTION or PROCEDURE name	103
Invalid key label	323
Invalid label file	14
Invalid MEMBER name	105
Invalid memory variable file	16
Invalid name character	71
Invalid order number	48
Invalid password	247
Invalid preprocessor identifier	321
Invalid printer redirection	196
Invalid PRO / FMO file	17
Invalid query file	18
Invalid report file	19
Invalid subscript reference	193
Invalid table name	320
Invalid value for convert size (8-24)	242
Invalid view file	21
Invalid window file	22
Key already exists in parent	334
Keyword Repeated	98
Loop stack overflow.	94
Macros cannot expand flow of control commands	153
Maximum #ifdef nesting exceeded	227
Maximum number of DO or UDF parameters exceeded	175
Maximum number of fields reached	36
Maximum number of locked records exceeded	222
Maximum number of nested FOR NEXT loops exceeded	172
MEMO field not allowed here	179
Memo file does not exist	337
Memory variable already defined - cannot make PUBLIC	190
Memory variable space exhausted	111
Menu already active	127
Method is not available on object	258
Missing ENDCASE	85
Missing ENDCLASS	89

Table F.2 Error codes in alphabetical order (continued)

Message	Code
Missing ENDDO	77
Missing ENDIF	75
Missing ENDPRINTJOB	87
Missing ENDSCAN	81
Missing ENDTEXT	91
Missing NEXT	79
Missing UNTIL	83
Must convert report before modifying	342
Nested views not allowed	162
NEXT encountered without preceding FOR	78
No bars defined for popup/pulldown	134
No fields were found to process	49
No matching parent record	332
No pads defined for	254
No PARAMETERS statement found	116
No previous DO WHILE/SCAN/PRINTJOB/FOR to match this command	219
No print driver selected	253
No Records Selected	259
No such bar	119
No such form	293
No such form object	126
No such listbox	214
No such menu	120
No such pad	121
No such popup	122
No such record in index	31
No such window	124
No such window	125
No table in use in area	34
Not a valid dBASE table	30
Not a valid table	325
Not an array	168
Not enough memory for DOS	112
Not enough memory for this operation	107
Not member of Class or Base class	289
Numbers are not allowed in the CURRENCY symbol	156
OLE Array is locked	363
OLE Bad call	366
OLE Bad parameter count	364
OLE Bad Variable Type	358
OLE BLOB field data is from an incompatible version	280
OLE BLOB field is corrupted	279

Table F.2 Error codes in alphabetical order (continued)

Message	Code
OLE Class name not in registry	371
OLE Data Type Mismatch	355
OLE Dispatch Exception	359
OLE Error	314
OLE field not allowed here	311
OLE Invalid Subscript	361
OLE Member Not found	353
OLE No Named arguments	357
OLE Not a collection	367
OLE Object does not support automation	369
OLE object error	278
OLE Overflow	360
OLE Parameter Not found	354
OLE Parameter not optional	365
OLE Unable to create object	370
OLE Unknown Class	362
OLE Unknown error	368
OLE Unknown interface	352
OLE Unknown name	356
OLE: cannot create link	295
only 1 dimensional array is allowed in this operation	315
only 1 or 2 dimensional array is allowed in this operation	283
Only allowed in function or procedure scope	324
Only valid in program files	115
Operation not allowed across different databases or table types	333
Operation not allowed for calculated fields	23
Operation not allowed in transaction	234
Operation not allowed on read-only fields	286
Operation not allowed on read-only files	24
Operation not allowed on read-only tables	287
Operation not allowed on this table	236
Out of range	164
Output parameter required	256
Parameter type '...' can only be used with CDECL calling convention	303
PARAMETERS command must be at top of function or procedure	70
Parent is not a PAGETEMPLATE or BANDBODY	308
Parent is not a REPORT	307
Popup already active	128
Popup too small	312
Port not configured for a printer	326
Position not in window	301
Preprocessor expansion too large	229

Table F.2 Error codes in alphabetical order (continued)

Message	Code
PRIMARY must start with first field	345
Printer error	189
Printer is either not connected or turned off	131
Procedure not found	182
Processing would exceed maximum allowed string length	181
Production index file does not exist	338
Program not SUSPENDed	118
Program too big to compile	106
Program was previously compiled with SET COVERAGE OFF	203
Property is not accessible	374
Property is read only	282
Property not found	285
Record is in use by another	109
Record is in use by another	373
Record out of range	5
Related records still exist in alias	329
Relation expression and active index expression must be the same	336
Relation using CONSTRAIN	331
Report Engine Error	284
Report writer has not been installed	347
Resource not found	265
Restricted command: not allowed in this context	199
Runtime buffer overflow	149
Selected tables cannot be related	317
Server Error	240
SET FIELDS space exhausted	218
SET KEY active in alias	330
Sharing buffers are full	223
Something is missing. Expecting	67
Stack overflow	148
Stored procedures not supported	261
String buffer overflow	150
Structure invalid	11
SUPER not allowed when THIS is undefined	290
Symbol table space exhausted. Increase to	217
Syntax error	63
Table already open	231
Table is not indexed	35
Tag not found	53
The system registry does not contain an OLE server for a file with extension	294
Too few arguments. Expecting at least	58
Too many arguments. Expecting at most	59

Table F.2 Error codes in alphabetical order (continued)

Message	Code
Too many dimensions	101
Too many files open	209
Too many index files	47
Too many nested DO/UDF	221
Too many nested expressions	161
Too many nested FOR loops.	95
Too many nested SCAN loops.	96
Too many RELATIONS in this chain	44
Too many symbols in this module	318
Too many UDF/PROCEDURES defined in program	102
UDF must return a value	100
UDF or procedure already exists	159
UDF or PROCEDURE not found	207
Unable to add data while constraints active	129
Unable to create control	292
Unable to execute DOS	160
Unable to open form	291
Unable to rename file	10
Unallowed phrase/keyword in command	97
Unauthorized access level	133
Unbalanced parentheses	62
Unexpected type	188
Unknown error saving window contents	274
Unknown keyword	68
Unmatched #else	225
Unmatched #endif	226
Unmatched parameters	117
Unrecognized command verb	54
Unterminated string	61
UNTIL encountered without preceding DO	82
Value out of range	192
Variable undefined	167
VBStream file Missing or Corrupt	349
VBX dlls cannot be RELEASED	348
Window not active	215
Window out of range	132
WindowMenu must be on child of MenuBar	33
Windows print file name longer than 31 characters	130
Work area already used in relation	42
Wrong version of IDAPI01.DLL	252

Index

Symbols

! command 41
 DOS vs. 224
 RUN vs. 467
operator 21
\$ operator 21
& (ampersand), comments 41
&& command 41
 * vs. 42
 NOTE vs. 42
* (asterisk)
 comment symbol 42, 43
 in fields 440
 wildcard character
 directory listings 9, 204,
 210
 fields list 516
 pattern matching 332
* command 42
 && vs. 42
* operator 20
** operator 20
+ operator 20
 concatenation 22
/ operator 20
:: operator 25
; (semicolon)
 command separator 519
 comment symbol 42
 continuation character 406
< operator 21
<= operator 21
<> operator 21
= operator 20, 21, 511
=< operator 21
== operator 21
=> operator 21
> operator 21
>= operator 21
? (question mark)
 temporary files 283
 wildcard character
 directory listings 9, 204,
 210
 fields list 516
 pattern matching 332
? command 43
 DEFINE BOX and 193
 ON PAGE and 380
 SET PRINTER and 547
 SET SPACE and 561
?? command 46
 ? command vs. 45
 ON PAGE and 380
 SET ALTERNATE and 480
 SET PRINTER and 547
 SET SPACE and 561
??? command 46
@...CLEAR 47
@...FILL 47
@...SAY...GET 47
 ON READERROR and 381
 SET CONSOLE and 489
 SET DEVICE and 504
 SET WINDOW OF MEMO
 and 568
 STORE AUTOMEM and 595
@...SCROLL 47
@...TO 48
^ operator 20
- operator 20
 concatenation 22

A

AbandonRecord() property 757
abbreviating keywords 7
 SET() and 569
ABS() 48
absolute values
 defined 48
 returning 48
accelerators 936
ACCEPT 48
ACCESS() 49
accessing
 alternate text editors 506
 arrays 274
 browse objects 760
 client/server
 applications 831, 951, 956
 OLE 796, 933
 data 131
 files 270
 multiuser
 environments 369
 file-sharing modes 513
 setting locks 265, 344,
 461
 read-only restrictions 516,
 532
 sequentially 274
 specific fields 514
files *See* low-level access
 commands
 functions 531
 procedures 531
ACOPY() 49
ACOS() 50
 RTOD() and 465
actions *See* events
ACTIVATE MENU 51
ACTIVATE POPUP 51
ACTIVATE SCREEN 52
ACTIVATE WINDOW 52
activating online Help 296, 521
activating the Debugger 185,
 506, 562
active indexes, returning 606
ActiveControl property 757
ADD clause 978
Add() property 759
adding fields 363, 978
adding records 70, 72, 309, 310,
 484, 766, 925, 983
 arrays and 75
 events and 857
 restricting 763
addition 111, 602, 611
addition operator 20
ADEL() 52
ADIR() 55
 Dir() vs. 793
Advise() property 760
 Unadvise() and 958
AELEMENT() 57
 AFILL() and 60
 ASUBSCRIPT() vs. 88
 Element() vs. 803
AFIELDS() 59
 Fields() vs. 808
AFILL() 60
 Fill() vs. 810
 STORE vs. 594
aggregation 111, 602, 611
AGROW() 61
 Grow() vs. 823
AINS() 64
 AGROW() vs. 62
ALEN() 67
alerts 125, 482
ALIAS clause 984
alias pointer 555
Alias property 760
ALIAS() 68
aliases
 automatically assigning 9

- catalogs 163
- defined 9
- field names 10, 441, 515
- indexes 528
- tables 760
 - linking 555
- work areas 9, 68, 477, 623
- _alignment 635
- _wrap and 663
- alignment
 - graphics 761
 - objects 836, 954
- Alignment property 761
- ALTER TABLE 978
- alternate text editors 164, 697
 - accessing 506
- alternate text files 479
- Alt-key combinations
 - See also* keyboard; keystrokes
 - command execution 375
- ampersand (&), comments 41
- Anchor property 763
- anchoring objects 763
- AND bitwise operator 97
- angles 395
 - arccosecant 84
 - arccosine 50
 - arccotangent 90
 - arcsecant 51
 - arcsine 84
 - arctangent 90, 91
 - converting
 - degrees to radians 226
 - radians to degrees 465
 - cosecant 578
 - cosine 157
 - cotangent 608
 - measuring 226
 - secant 157
 - sine 577
 - tangent 608
- ANSI conversions 69
- ANSI date format 496
- ANSI() 69
 - OEM() and 370
- Answer tables 588
 - See also* Paradox tables
- _app 636
- APPEND 70
 - APPEND AUTOMEM vs. 72
 - INSERT vs. 309
 - KEYMATCH() and 325
 - SET CARRY and 484
- APPEND AUTOMEM 72
 - CLEAR AUTOMEM and 130
 - INSERT AUTOMEM vs. 311
 - SET CARRY and 484
- APPEND BLANK 71
 - BLANK vs. 102
 - SET CARRY and 484
- APPEND FROM 73
- APPEND FROM ARRAY 75
- APPEND MEMO 77
- Append property 763
- application object 636
- applications
 - See also* programs
 - client/server *See* DDE server
 - applications; OLE server applications
 - copying 388
 - external 693, 796, 831, 919
 - MDI 841, 842
 - multiuser *See* multiuser environments
 - sound 396, 796
 - standalone 573, 707
 - Windows, running 467
- arccosecant, returning 84
- arccosine, returning 50
- arccotangent, returning 90
- arcsecant, returning 51
- arcsine, returning 84
- arctangent, returning 90, 91
- ARESIZE() 78
 - Resize() vs. 923
- arguments 7, 29
 - See also* parameters
- automem variables 442
- color 194
- file-name skeletons 11
- arithmetic operations
 - arithmetic mean, returning 92, 111
 - remainders, returning 360
 - type conversions 625
- array elements 11, 187
 - adding to arrays 61, 759, 823, 832, 923
 - copying 49
 - counting 57, 803, 940
 - deleting 52, 790, 922
 - numbers 187
 - referencing 87, 948
 - returning 67, 78
 - sorting 84, 941
 - subscripts 187, 922, 923
 - adding 759, 823
 - deleting 790
 - finding 87, 812, 833, 852, 948
- array objects 679, 681
- arrays 11, 75, 792
 - accessing 274
- adding elements 61, 823, 832, 923
- assigning values 60, 64, 188, 810
- copying data 154, 445
- declaring 187, 412
- deleting 431
- deleting elements 922
- expressions
 - finding 82, 927
 - storing 593
- file information 55, 793, 794
- initializing 60, 596
- LOCAL and 341
- maximums and limits 1003
- memo fields 447, 596
- multi-dimensional, creating 62
- passing as parameters 404
- size, changing 61, 78
- storing values 93, 112
- table structures 59, 808
- two-dimensional, creating 62
- updating 79
- arrow keys
 - See also* keyboard; keystrokes
- assigning
 - command execution 376
 - input focus 822
- ASC() 81
 - CHR() vs. 124
- ASCAN() 82
 - Scan() vs. 927
- ascending sort order 304, 582
- ASCII chart 1013
- ASCII text files *See* text files
- ASCII values
 - See also* decimal values
 - converting to characters 124
 - returning 81
- ASIN() 84
 - RTOD() and 465
- ASORT() 84
 - Sort() vs. 941
- assigning keystrokes
 - command execution 372, 375, 519, 526
 - interrupts 510
- assignment 4
- assignment operator 20
- asterisk (*)
 - comment symbol 42, 43
 - in fields 440
 - wildcard character
 - directory listings 9, 204, 210
 - fields list 516

- pattern matching 332
- ASUBSCRIPT() 87
 - AELEMENT() vs. 58
 - Subscript() vs. 948
- AT() 89
 - RAT() and 420
- ATAN() 90
 - ATN2() vs. 91
 - RTOD() and 465
- ATN2() 91
 - ATAN() vs. 90
 - RTOD() and 465
- attributes
 - color codes 778
 - file (DOS) 56, 248, 270, 793
 - file (Windows95) 794
 - fonts 815, 818, 819
 - objects *See* properties
 - text 44
 - size 817, 926
- automatic backup 205
- automatic compiling 503
- automatic file locks 265, 339, 461
 - disabling 532
- automatically saving data 481
- automatically updating
 - indexes 305
- autotem variables
 - See also* memory variables
 - arguments 442
 - clearing 129, 433
 - creating 129, 595
 - defined 72
 - storing data 595
- AutoSize property 764
- AVERAGE 92
 - SET HEADINGS and 520
- averages, returning 92, 111

B

- backgrounds
 - blended (hatched) 106, 229, 906
 - colors 778
 - setting 106, 229
 - high intensity 778
- backup files
 - changing tables 362
 - disk space, checking 205
- backward compatibility 989
 - See also* dBASE IV commands
- BAR() 93
- BARCOUNT() 93
- BARPROMPT() 94
- BDE *See* IDAPI
- beeps 125, 482
- Before property 765
- BeginAppend() property 766
- beginning-of-file indicator 103
- BEGINTRANS() 94
- BELL parameter 482
 - See also* alerts
- benchmarks 235, 472
- binary data types 15
 - combining 146
 - user-defined 145
- binary fields 77
 - changing 443
 - copying 145
 - data type, returning 96
 - sound effects 395
- binary files
 - coverage analysis 490
 - creating 145
 - reading from 443
- binary operators 20
- BINTYPE() 96
- BITAND() 97
- BITLSHIFT() 98
- bitmaps 145
 - See also* graphics
 - pushbuttons 795, 813, 960
- BITOR() 99
- BITRSHIFT() 99
- BITSET() 100
 - OnLeftDbClick and 870
 - OnLeftMouseDown and 872
 - OnLeftMouseUp and 873, 895
 - OnMiddleDbClick and 876
 - OnMiddleMouseDown and 878
 - OnMiddleMouseUp and 880
 - OnRightDbClick and 891
 - OnRightMouseDown and 893
- bitwise operators 100
 - AND 97
 - OR 99
 - shift bits 98, 99
 - XOR 101
- BITXOR() 101
- BLANK 102
- blank dates 14, 225
- blank expressions *See* EMPTY(); ISBLANK()
- blank fields 102, 260
- blank records 71, 309
 - averaging numeric fields 93
- blank values 102
 - calculations 112

- blended (hatched)
 - backgrounds 106, 229, 906
- BOF() 103
 - RECNO() and 425
 - SKIP and 579
- bold type 44, 815
- bookmark data types 105
- BOOKMARK() 104
 - GO vs. 295
- bookmarks 15, 294
 - See also* record pointers
 - returning 104
- Boolean expressions *See* logical expressions
- Border property 770
- borders 740, 745, 909, 910
 - adding to objects 770, 771
 - disabling 771
 - forms, setting 836, 954
- BorderStyle property 771
 - Border vs. 771
- Bottom property 772
 - Right and 924
- _box 638
- boxes, drawing 193
- branching commands *See* loops
- breakpoints
 - See also* debugging
 - defined 603
- British date format 496
- BROWSE 105
 - EDIT vs. 232
 - exiting 109
 - SET CARRY and 484
 - SET REFRESH and 552
- browse objects 684, 824
 - accessing tables 761
 - changing data 845
 - key fields 814
 - delete boxes 936
 - deleting records 789, 936
 - displaying data 807, 844, 953
 - losing data 814
 - record number column 938
 - restricting data entry 763
 - viewing field names 937
- BROWSE window *See* Table Editor
- browsing 230
- buffers
 - data
 - updating 428
 - writing to disk 268
 - file, flushing 255
 - typeahead 324
 - clearing 134

- information, getting 306, 369
- size, setting 566

BUILD 110

built-in classes 5

built-in functions 4

C

C calling conventions 242

CALCULATE 111

CALCULATE AVG() 111

CALCULATE CNT() 111

CALCULATE MAX() 111

CALCULATE MIN() 111

CALCULATE NPV() 111

CALCULATE STD() 111

CALCULATE SUM() 111

CALCULATE VAR() 111

calculated fields

- accessing 516
- displaying 108, 231, 808
- editing 232
- indexing 305

call chain 218

- preprocessor 674

call operator 23

call stack (defined) 458

calls *See* function calls; procedure calls

CANCEL 113

- QUIT vs. 417
- RETURN vs. 459
- SUSPEND vs. 603

canceling program

- execution 113

CanClose property 773

capital gains 284

- present value 415

capitalization 408

- See also* uppercase letters

carriage returns

- automatic 972
- character, counting 328, 330, 460
- substrings 89, 601
- files 256

case

- See also* lowercase letters; uppercase letters
- converting 304, 348, 620
- first letter 408
- testing 314, 317, 320

case sensitivity 351, 358

- program code 28
- searches 82, 89, 420, 928
- pattern matching 332

- sorting data 582

catalog files 564

- adding new files 486
- creating 162
- names, returning 114
- opening 485
- updating 486

CATALOG() 114

CATALOG.CAT 486

catalogs, defined 162

CD 115

- RUN and 467
- SET DIRECTORY vs. 505

CDOW() 116

CEILING() 117

- compared 313

CENTER() 118

centering graphics 762

centering text 118

century 487

CENTURY parameter 487

CERROR() 120

CHANGE 121

- EDIT vs. 233
- SET REFRESH and 552

CHANGE window *See* Table Editor

CHANGE() 122

- CONVERT and 142

changes, undoing 802, 959

- multiuser environments 463

changing

- See also* editing
- binary fields 443
- data 404, 429, 439, 619
- automatically 785
- browse objects 814, 845
- DDE applications 760, 854, 856, 958
- multiple fields 440
- multiuser environments 122, 265, 344, 461
- specific records 440
- data types 362
- drive and directory 9, 115
- current working 504
- fields

 - names 362
 - widths 362

- file names 437
- fonts 45
- forms 161, 165, 171
- key fields 440

 - browse objects 814

- memo fields 440, 447
- mouse pointers 846, 882

- numeric fields 440
- objects, definitions 427
- property settings 312
- SET command values 478
- tables 160

 - names 438
 - structures 161, 362

- text, spin boxes 945

character codes (data types) 59, 809

character constants 13

character data

- case, testing 314, 317, 320
- converting dates 225, 227
- converting numbers 363, 598
- deleting specific characters 599
- finding 474, 475
- DLLs 452
- formatting 612
- key expressions 304
- returning logical 363
- returning numbers 363, 625

character data types 13

character expressions

- assigning to keystrokes 519
- deleting spaces 349, 466, 613
- phonetic values 203, 584
- picture templates 43, 612
- repeating 450
- writing to files 286

character fields 13, 141, 440

- display widths 109, 232
- indexing 630
- structure-extended tables 178
- text files 144

character sets 529, 530

- converting 69
- current, returning 123

character strings *See* strings

character types 597

characters

- ASCII values 81
- returning from 124
- carriage return 256, 276
- continuation lines 406
- converting to dates 363
- currency symbol 493
- date separators 535, 580
- changing 496
- decimal separator 545
- function templates 44, 820
- linefeeds 256, 276
- literal 332
- null 330
- padding 119
- pattern matching 332

- program comments 42
- space 586
- thousands separator 559
- time separators 580
- wildcard
 - directory listings 9, 204, 210
 - fields list 516
 - pattern matching 332
 - temporary files 283
- CHARSET() 123
- check boxes 686
 - displaying 855
- Checked property 774
- checkmarks, adding to
 - menus 774
- child tables 554
 - moving through 560
- CHOOSEPRINTER()
 - _pdriver and 649
 - _porientation and 656
 - _ppitch and 657
- choosing
 - See also* selecting
 - menu commands 936
- CHR() 124
 - ASC() vs. 81
 - SET BELL and 482
- circles 394
- circular functions *See*
 - trigonometric functions
- CLASS ARRAY 679
- CLASS ASSOCARRAY 681
- CLASS BROWSE 682
- CLASS CHECKBOX 686
- CLASS COMBOBOX 690
- CLASS DDELINK 693
- CLASS EDITOR 697
- CLASS ENTRYFIELD 700
- CLASS FORM 704
- CLASS IMAGE 708
- CLASS LINE 710
- CLASS LISTBOX 712
- CLASS MENU 716
- CLASS MENUBAR 719
- CLASS OBJECT 721
- CLASS OLE 721
- CLASS OLEAUTOCLIENT 725
- CLASS PAINTBOX 726
- CLASS POPUP 729
- CLASS PUSHBUTTON 731
- CLASS RADIOBUTTON 735
- CLASS RECTANGLE 739
- CLASS SCROLLBAR 742
- CLASS SHAPE 745
- CLASS SPINBOX 746
- CLASS TABBOX 749
- CLASS TEXT 752
- CLASS...ENDCLASS 125
- classes *See* object classes
- ClassName property 775
- CLEAR 128
- CLEAR ALL
 - RELEASE AUTOMEM and 433
- CLEAR AUTOMEM 129
 - RELEASE AUTOMEM and 433
 - STORE AUTOMEM vs. 595
- CLEAR FIELDS 131
- CLEAR GETS 131
- CLEAR MEMORY 131
- CLEAR MENUS 132
- CLEAR POPUPS 133
- CLEAR PROGRAM 133
- CLEAR SCREENS 134
- CLEAR TYPEAHEAD 134
- CLEAR WINDOWS 134
- clearing memory
 - object definitions 921
 - unallocated 268
- clearing memory variables 131, 432, 453
 - non-public 458
 - program execution and 113, 417
 - system 5
- clearing typeahead buffers 134
- client/server applications
 - See also* DDE server
 - applications; OLE server applications
 - MDI forms 841, 842
- clock 181, 610
 - See also* time
 - setting 497, 563
 - time elapsed 235, 472
- CLOSE 135
- CLOSE ALL 135
 - FLUSH and 268
- CLOSE ALTERNATE 135
 - SET ALTERNATE and 480
- CLOSE DATABASES 135
 - FLUSH and 268
- CLOSE FORM
 - READMODAL() and 422
- CLOSE FORMAT 135
- CLOSE FORMS 135
- CLOSE INDEXES 135
 - FLUSH and 268
- CLOSE PRINTER 135
- CLOSE PROCEDURE 135
- CLOSE TABLES 135
- Close() property 776
- closing
 - databases 135
 - files 135, 247, 417
 - text 135
 - forms 135, 773, 776, 805, 862
 - indexes 135
 - procedures 135
 - program files 113, 549
 - tables 135, 136
 - work areas 135
- CMONTH() 137
- code
 - See also* program files
 - adding comments 41, 42
 - temporarily 43
 - case sensitivity 28
 - compiling *See* compiling
 - constructor 126
 - coverage analysis 675
 - editing 140, 504
 - optimizing 669
 - protecting 140
 - repeating statements *See*
 - loops
 - testing 120, 490
- code pages 1007
- codeblocks 17–19
 - calling 23
 - executing, forms 898
- COL() 137
- collation values
 - See also* language drivers
 - comparing 351, 358
- color and attribute codes 778
- color arguments 194
- color palettes 288
- ColorHighlight property 777
- ColorNormal property 745, 777
 - ColorHighlight vs. 777
- colors
 - background 106, 229
 - defining 106, 194, 229
 - custom 288
 - dBASE IV windows 488
 - objects 777
 - redefining 194
 - selecting 288
- colors commands
 - DEFINE COLOR 194
 - GETCOLOR() 288
 - ISCOLOR() 317
 - SET COLOR OF 488
 - SET COLOR TO 488
- columns *See* fields
- combo boxes 690, 799, 947

- displaying data 786
- prompts 942
- command processing 458
 - See also* program execution
- Command window
 - clearing results pane 128
 - colors, setting 488
 - coverage analysis 209, 490
 - creating files 564
 - debugging programs 186
 - displaying files 209, 614
 - displaying messages 211, 562
 - current environment 212
 - entering expressions 291
 - maximums and limits 1003
 - pausing program
 - execution 580
 - restoring memory
 - variables 454
 - resuming program
 - execution 456
 - returning table structures 214
 - saving output 479
 - shelling to DOS 573
 - suspending program
 - execution 603
 - writing to 206, 334, 489, 548
- commands 3-4, 479
 - See also* specific dBASE
 - commands
 - abbreviating 7
 - executing
 - page formatting 379
 - shortcuts 372, 519, 526
 - implicit 4
 - menu *See* menu commands
 - comments 41, 42
 - temporary 43
 - COMMIT() 138
 - committing transactions 138
 - DDE applications 953
 - common logarithms 345
 - comparing
 - dates 350, 358
 - expressions 350, 357
 - float values 117, 267
 - logical expressions 350, 357
 - multi-table records 322
 - numeric data, equality 117, 267
 - record counters 122
 - strings *See* string comparisons
 - comparison operator (=) 511
 - COMPILE 139
 - compiler errors 120
 - compiler *See* preprocessor
 - compiling 5, 667, 675, 676
 - automatically 503
 - canceling 140
 - conditional 668, 670, 671, 672
 - format files 503, 518
 - multiple programs 674
 - options, setting 675
 - specified files 140
 - unrelated files 140
 - compound index files *See* .MDX files
 - concatenation operator 22
 - concatenation, date fields 227
 - conditional execution 219, 299, 301
 - OS() 388
 - VERS() 627
 - conditions 470
 - evaluating 299
 - events 804, 967
 - exceptions 219
 - search operations 342
 - setting 516, 668, 961, 968
 - testing 272, 300
 - multiple 300
 - cones, measuring volume 395
 - confirmation messages 558
 - connecting tables *See* linking and relating
 - constants
 - See also* numbers
 - changing 667, 669
 - character 13
 - identifiers 668
 - pi 395
 - constructor code 126
 - context-sensitive help 565, 827
 - continuation lines
 - comments 42
 - procedures 406
 - CONTINUE 140
 - EOF() and 238
 - FIND vs. 260
 - FOUND() and 274
 - LOCATE and 343
 - continuing search
 - operations 140
 - control codes (printer) 544
 - control keys *See* Ctrl keys
 - Control menu 949
 - control structures
 - linear 219, 299, 301
 - loops 220, 222, 272, 470
 - controlling table access 409
 - controls *See* objects
 - CONVERT 141
 - COPY and 144
 - COPY STRUCTURE and 150
 - converting
 - ASCII values to
 - characters 124
 - case 304, 348, 620
 - first letter 408
 - characters to ASCII 81
 - characters to dates 179
 - characters to numbers 363, 625
 - dates to characters 225, 227
 - dates to strings 216, 353
 - decimal to hexadecimal 320
 - degrees to radians 226
 - external functions 244
 - hexadecimal to decimal 297
 - incompatible data types 363
 - index files to tags 147, 152
 - logical fields to
 - characters 363
 - memo fields to character 440
 - numbers to characters 363, 598
 - numbers to logical values 363
 - radians to degrees 465
 - strings *See* string conversions
 - coordinates *See* screen coordinates
 - COPY 143
 - PACK and 389
 - SET ESCAPE and 510
 - COPY BINARY 145
 - COPY FILE 146
 - COPY INDEXES 147
 - See also* COPY TAG
 - COPY MEMO 148
 - COPY STRUCTURE 150
 - COPY TABLE 151
 - COPY TAG 152
 - See also* COPY INDEXES
 - COPY TO ARRAY 153
 - COPY TO clause 985
 - COPY TO...STRUCTURE
 - EXTENDED 155
 - AFIELDS() vs. 59
 - CREATE...FROM and 175
 - CREATE...STRUCTURE
 - EXTENDED vs. 178
 - Fields() vs. 809
 - Copy() property 779
 - copying
 - applications 388
 - array elements 49
 - binary fields 145
 - data 71, 73
 - arrays and 153, 445
 - automatically 143
 - multiple fields 144, 150

- to specific records 484
- files 146, 626
- index files 144, 152
- memo fields 76, 144, 148
 - to text files 144
- memory variables 453
- tables 143, 151
 - structures 150, 155, 176
- text 779, 800
- text files 77, 448
- COS() 157
 - ACOS() and 51
 - DTOR() and 226
 - PI() and 395
- cosecant 578
 - inverse 84
- cosine 157
 - inverse, returning 50
 - reciprocal 157
- cotangent 608
 - inverse 90
- COUNT 159
 - RECCOUNT() vs. 424
- Count() property 782
 - Selected() and 931
- counting array elements 57, 803, 940
- counting fields 262
- counting records 111, 204, 210, 424
- coverage files 208
 - creating 490
 - updating 490
- CREATE 160
 - CREATE...FROM vs. 176
- CREATE... commands
 - SET DESIGN and 502
- Create an Expression dialog box 291
- CREATE APPLICATION 161
- CREATE CATALOG 162
- CREATE COMMAND 164
- CREATE FILE 165
- CREATE FORM 165
- CREATE INDEX 979
- CREATE LABEL 167
 - LABEL FORM and 326
- CREATE MENU 168
- CREATE POPUP 168
- CREATE QUERY 169
 - CREATE VIEW vs. 174
- CREATE REPORT 169
 - REPORT FORM and 452
- CREATE SCREEN 171
- CREATE SESSION 171
- CREATE TABLE 979

- CREATE VIEW 174
 - CREATE QUERY vs. 169
- CREATE VIEW...FROM ENVIRONMENT 174
- CREATE...FROM 175
 - COPY TO...STRUCTURE EXTENDED and 156
 - CREATE...STRUCTURE EXTENDED and 178
- CREATE...STRUCTURE EXTENDED 177
 - CREATE...FROM and 175
- CTOD() 179
 - SEEK and 474
 - SEEK() and 475
- Ctrl keys
 - See also* keyboard; keystrokes
 - command execution 375, 519
- CUATab property 783
 - _curobj 639
- CURRENCY parameter 494
- currency symbols 493
- current database 495
 - name, returning 180
- current date
 - returning 181
 - setting 497
- current form 386, 934
- current object 422
 - colors, setting 777
 - finding 757
- current record
 - number, returning 425
 - updating 442
- current settings 569, 571
 - character set 123
 - environment 212
 - language driver 327
- current work areas 629
- current working directory *See* directories
- current working drive *See* disk drives
- current working environment 175
 - See also* views; work areas
- CurSel property 784
- cursors
 - controlling 492, 783
 - hiding 494
 - moving 489
- custom classes 126
- custom editing controls 726
- custom formats
 - See also* format files
- custom Help 866

- Cut() property 785
- cylinders, measuring volume 395

D

- data
 - accessing 131
 - files 270
 - multiuser environments 369
 - file-sharing modes 513
 - setting locks 265, 344, 461
 - read-only restrictions 516, 532
 - sequentially 274
 - specific fields 514
 - changes, undoing, multiuser environments 463
 - changing 404, 429, 439, 619
 - automatically 785
 - browse objects 814, 845
 - DDE applications 760, 854, 856, 958
 - multiple fields 440
 - multiuser environments 122, 265, 344, 461
 - specific records 440
 - converting *See* converting
 - copying 71, 73
 - arrays and 153, 445
 - automatically 143
 - multiple fields 144, 150
 - to specific records 484
 - deleting *See* deleting
 - displaying 45, 489, 526
 - compressing records 229
 - memo fields 232, 538
 - objects and 786, 807, 844, 953
 - specific records 206
 - with BROWSE 105
 - with CHANGE 121
 - editing *See* deleting
 - exchanging *See* DDE links
 - filtering 117
 - setting filters 517
 - formatting *See* formats
 - losing 255, 362, 389, 440
 - browse objects 814
 - minimizing loss 481
 - manipulating 117, 581
 - organizing 304, 583
 - overwriting 440
 - binary fields 145
 - confirmation messages 558

- memo fields 77, 149, 447, 448
- printing *See* printing
- processing 11, 375
 - optimizing 481
 - specific records 516, 527
 - specified ranges 526
- protecting 409, 502
- retrieving 984
- sample 288
- saving 268
 - automatically 481
 - multiuser environments 138
- searching *See* searching
- similar spellings, finding 584
- sorting *See* sorting data
- updating 350, 619, 985
- validating 963

data buffers 268

updating 428

data entry

See also entry fields

- controlling 72, 595, 961
- cursors, moving 489
- DDE applications 912
- DO...UNTIL and 223
- errors, trapping 381
- invalid 482, 962
- restricting 502, 763, 875
- templates 820, 911
- validating 72, 595

data integrity 268

data types 12–19, 1006

- arrays and 50
- binary fields 96, 145
- blank values 102
- bookmark 105
- changing 362
- character codes 59, 809
- conversions *See* converting; type conversions
- DLLs 242
- external functions 244
- finding 324
- incompatible 363
- local SQL 980
- memo fields 597
- returning 615
- user-defined 145

database servers

See also DDE server

- applications; OLE server applications
- connecting to 385, 796, 831, 919
- disconnecting 951

DATABASE() 180

databases 9

See also tables

- closing 135
 - associated files 135
- current, specifying 495
- maximums and limits 1001
- names, returning 180
- opening 385
- SQL *See* SQL databases
- transactions, rolling back 138, 463

DataLink property 785

DataSource vs. 787

SelectAll and 930

DataSource property 786

DataLink vs. 786

date and time commands

- CDOW() 116
- CMONTH() 137
- DATE() 181
- DAY() 181
- DMY() 216
- DOW() 224
- ELAPSED() 235
- MDY() 353
- MONTH() 364
- SECONDS() 472
- SET CENTURY 487
- SET DATE 496
- SET DATE TO 497
- SET MARK 535
- SET TIME 563
- TIME() 610
- YEAR() 630

date and time stamps 497, 563

returning 251, 281

date data types 14

date fields

- concatenating 227
- converting characters 363
- indexing 227, 630

date formats 487

- current 116
- returning 216, 225, 227, 353
- SLEEP 580
- specifying 496

DATE parameter 496

DATE() 181

- DTOC() and 225
- SET DATE TO and 497

dates 496, 946

- blank 14, 225
- comparing 350, 358
- converting to characters 225, 227
- converting to strings 216, 353
- default settings 496, 535

finding 474, 475

formatting 612

invalid 381

key expressions 304

literal 14

manipulating 179, 225

resetting 497

returning 137, 181, 350, 364

- character expressions as 179
- system 181
- weekdays 116, 224
- year 487, 630

separators 535, 580

- changing 496

sorting 227

subtracting 20

valid range 498

DAY() 181

dBASE

- data types 980
- exiting 417
- home directory 640
- maximums and limits 1001
- online help *See* Help system
- version numbers, returning 627

dBASE III PLUS files 143, 174, 517

dBASE IV commands (backward compatible)

- @...CLEAR 47
- @...FILL 47
- @...SAY...GET 47
- @...SCROLL 47
- @...TO 48
- ACCEPT 48
- ACTIVATE SCREEN 52
- CLEAR GETS 131
- CLEAR SCREENS 134
- CLEAR TYPEAHEAD 134
- COL() 138
- FUNCTION 282
- INPUT 307
- ISCOLOR() 317
- KEYBOARD 324
- LASTKEY() 327
- MCOL() 351
- MDOWN() 352
- MROW() 365
- ON MOUSE 378
- PARAMETERS 391
- READ 421
- READKEY() 421
- RELEASE SCREENS 436
- RESTORE SCREEN 456
- ROW() 465
- SAVE SCREEN 470

- SET BORDER 484
- SET COLOR OF 488
- SET COLOR TO 488
- SET DEFAULT 501
- SET DELIMITERS 502
- SET DEVICE 504
- SET DISPLAY 506
- SET ECHO 506
- SET FORMAT 518
- SET INTENSITY 526
- SET MESSAGE 539
- SET MOUSE 539
- SET ODOMETER 540
- SET STEP 562
- SET TYPEAHEAD 566
- TEXT 610
- UPDATED() 620
- VARREAD() 627
- dBASE IV menu commands
 - ACTIVATE MENU 51
 - ACTIVATE POPUP 51
 - BAR() 93
 - BARCOUNT() 93
 - BARPROMPT() 94
 - CLEAR MENUS 132
 - CLEAR POPUPS 133
 - DEACTIVATE MENU 184
 - DEACTIVATE POPUP 185
 - DEFINE BAR 193
 - DEFINE MENU 195
 - DEFINE PAD 195
 - DEFINE POPUP 196
 - MENU() 356
 - ON BAR 371
 - ON EXIT BAR 374
 - ON EXIT MENU 374
 - ON EXIT PAD 374
 - ON EXIT POPUP 374
 - ON MENU 377
 - ON PAD 379
 - ON POPUP 381
 - ON SELECTION BAR 382
 - ON SELECTION MENU 384
 - ON SELECTION PAD 384
 - ON SELECTION POPUP 384
 - PAD() 390
 - PADPROMPT() 390
 - POPUP() 397
 - PROMPT() 407
 - RELEASE MENUS 435
 - RELEASE POPUPS 436
 - SHOW MENU 574
 - SHOW POPUP 576
- dBASE IV printing commands
 - ??? command 46
 - _alignment 635
 - _box 638
 - DEFINE BOX 193
 - _indent 640
 - _lmargin 642
 - _padvance 643
 - _pageno 644
 - _pbpage 645
 - _pcolno 646
 - _pcopies 647
 - _pdriver 648
 - _pject 649
 - _pepage 650
 - _pform 651
 - _plength 653
 - _plineno 654
 - _ploffset 656
 - _porientation 656
 - _ppitch 657
 - _pquality 658
 - _pspacing 659
 - _rmargin 661
 - _tabs 662
 - _wrap 663
- dBASE IV SQL commands 977–986
- dBASE IV windows commands
 - ACTIVATE WINDOW 52
 - CLEAR WINDOWS 134
 - DEACTIVATE WINDOW 185
 - DEFINE WINDOW 196
 - MOVE WINDOW 365
 - RELEASE WINDOWS 437
 - RESTORE WINDOW 456
 - SAVE WINDOW 470
 - WINDOW() 629
- _dbaselock fields 122, 141, 144
 - accessing 339
 - copying 150, 156
- _dbasewin
 - #ifdef and 671
- DBASEWIN directory 640
- DBASEWIN.EXE
 - path, returning 297
- DBASEWIN.EXE directory 297
- DBASEWIN.INI, changing
 - settings 478
- DBERROR() 182
- DBF() 183
- DBMESSAGE() 184
- _dbwinhome 640
- DDE links
 - disabling 951
 - information, getting 932, 955
 - setting 831
- DDE server applications 693
 - accessing 831, 951, 956
 - changing data 854, 856, 958
- events, trapping 863, 886, 889, 890, 901
 - name, returning 932, 955
 - reading from 907
 - transactions, committing 953
 - writing to 806, 912
- DdeServiceName 636
- DEACTIVATE MENU 184
- DEACTIVATE POPUP 185
- DEACTIVATE WINDOW 185
- DEBUG 185
- Debugger 506, 562
 - opening 185
- debugging 670
 - comments, temporary 43
 - coverage analysis 208, 490
 - procedures 185, 186, 406
 - program flow, tracking 333
 - programs, stepping through 186
 - records, stepping through 470
 - suspending program execution 603
 - UDFs 185, 186, 406
- debugging commands
 - See also* error handling
 - DEBUG 185
 - DISPLAY COVERAGE 208
 - GENERATE 288
 - LIST COVERAGE 334
 - RESUME 456
 - SET COVERAGE 490
 - SET ECHO 506
 - SET STEP 562
 - SUSPEND 603
- decimal digits 598
 - decimal separator 545
 - deleting 313
 - equality 117, 267
- decimal places 500
 - returning 252, 464
- decimal values
 - See also* ASCII values
 - converting to hexadecimal 320
 - keystrokes, returning 306, 369
 - returning 297
 - random numbers 417
- DECIMALS parameter 500
- declarations
 - arrays 187, 412
 - external functions 244
 - object classes 125
 - procedures 401
 - UDFs 282

- variables 412
 - local 341
 - private 399
 - static 591
- DECLARE 187
 - STORE MEMO and 596
- decreasing spin box values 946
- Default property 788
- defaults
 - date and time 496, 535
 - separators 580
 - decimal places 500
 - file names 143
 - forms 71, 765
 - function keys 519
 - sort order 582
 - system bell 482
 - tables 499
 - working directory 505
- DEFINE 189
 - Left and 836
 - REDEFINE and 427
 - Top and 955
- #define 667
- DEFINE BAR 193
- DEFINE BOX 193
 - _box and 638
 - _pspacing and 660
- DEFINE COLOR 194
- DEFINE MENU 195
- DEFINE PAD 195
- DEFINE POPUP 196
- DEFINE WINDOW 196
- defining fields lists 514
- degrees
 - converting to radians 226
 - returning 465
- delaying program execution 580
- DELETE 196
 - PACK vs. 389
 - RECALL and 423
- DELETE ALL
 - ZAP vs. 630
- DELETE FILE 198
 - ERASE vs. 239
- DELETE FROM 981
- Delete property 789
- DELETE TABLE 198
- DELETE TAG 199
- Delete() property 790
- DELETED() 200
- deleting
 - See also* erasing
 - arrays 431
 - elements 52, 790, 922
 - decimal digits 313
 - fields 363, 978
 - files 198, 239
 - index files 199, 983
 - index tags 199
 - indexes 982
 - leading spaces 349
 - memo files 983
 - memory variables 431
 - records 196, 200, 389, 630, 757, 981
 - confirming 558
 - controlling 501
 - marking prevented 108, 231, 789, 936
 - specified characters 599
 - tables 198, 982
 - text 785, 801
 - trailing spaces 466, 613
 - trailing zeros 464
- delimited text files 144
- delimiters
 - command execution 519
 - date 535, 580
 - changing 496
 - directory paths 542
 - nested strings 13
 - thousands 559
 - time 580
- delineating output 193
- derived classes 6
- descending sort order 304, 582
- DESCENDING() 201
- designing forms 791
- designing table structures 177
- DesignView property 791
- desktop, aligning objects 836, 954
- development tools 490
- dialog box objects *See* check boxes; entry fields; radio buttons
- dialog boxes 365, 422, 918
- DIFFERENCE() 202
 - LIKE() vs. 332
 - SOUNDEX() and 584
- Dimensions property 792
- Dir() property 793
- DIR/DIRECTORY 203
 - SET SEPARATOR and 559
- directives *See* preprocessor directives
- directories
 - changing 9, 115
 - current working 504
 - creating 352, 359
 - dBASE home 640
 - searching 543
- directory lists 203, 210
- directory paths
 - See also* search path
 - returning 407, 518
 - DBASEWIN.EXE 297
 - separators 542
- DirExt() property 794
- DisabledBitmap property 795
- disk and file management *See* file management commands
- disk and file utilities *See* file utilities and information; system utilities and information
- disk drives
 - changing 115
 - current working 504
 - disk space, returning 204, 205, 210
 - freeing memory *See* memory
 - invalid 505
 - valid, returning 626
- disk files *See* files
- DISKSPACE() 205
- DISPLAY 206
 - ? command vs. 45
 - SET HEADINGS and 520
 - TRANSFORM() and 613
- DISPLAY COVERAGE 208
- DISPLAY FILES 209
 - SET SEPARATOR and 559
- DISPLAY MEMORY 211
 - LOCAL and 341
 - PRIVATE and 399
- display options 107, 230
 - See also* fonts
 - browse objects 807, 953
- DISPLAY STATUS 212
 - IMPORT and 302
- DISPLAY STRUCTURE 214
 - IMPORT and 302
 - RECSIZE() and 426
- display widths
 - character fields 109, 232
 - numeric fields 331
- displaying
 - See also* viewing
 - check boxes 855
 - data 45, 489, 526
 - compressing records 229
 - field names 520
 - memo fields 232, 538
 - objects and 786, 807, 844, 953
 - specific records 206
 - with BROWSE 105
 - with CHANGE 121
 - editing windows 164

- field definitions 214
- forms 971
- graphics 454, 709
- labels 326
- messages 489, 562, 580
 - current environment 212
 - in status bars 539, 945, 962
 - memory variables 211
- prompts 787
- radio buttons 855
- reports 452
- table structures 214
- text files 614
- windows 580
- displays
 - See also* screens
 - color attributes 778
 - intensity, setting 778
 - mode, setting 506
 - slowing 235
- DISTINCT keyword 984
- division operator 20
- division, remainders,
 - returning 360
- DLLs 813
 - character strings, getting 452
 - defined 340
 - files, search path 243
 - initializing 340
 - prototype functions 242, 637
 - releasing 434
 - return values 97, 98, 99, 100, 101
- DMY() 216
- DO 217
 - COMPILE vs. 140
 - SET DEVELOPMENT and 503
 - SUSPEND and 604
- DO CASE 219
 - IF vs. 300
- DO WHILE 220
 - DO CASE and 220
 - DO...UNTIL vs. 223
 - SCAN vs. 471
 - SLEEP vs. 580
- DO...UNTIL 222
 - DO WHILE vs. 221
- documentation, printing
 - conventions 2, 7
- documenting programs 41, 42
- DOS
 - commands, executing 41, 223, 467
 - environment variables 290
 - file attributes 56, 248, 270, 793
 - return codes 417
- DOS command 223
 - RUN vs. 467
 - RUN() vs. 468
- DoVerb() property 796
- DOW() 224
- DownBitmap property 798
- drawing lines 710, 908
- drawing shapes 909, 910, 935
- drives *See* disk drives
- DROP clause 978
- DROP INDEX 982
- DROP TABLE 982
- DropDownHeight property 799
- DTOC() 225
 - DTOS() vs. 227
- DTOR() 226
 - COS() and 157
 - SIN() and 577
- DTOS() 227
- duplicate values 325
 - keys 560
- duplicating character strings 450
- duration (bell) 482
- dynamic-link libraries *See* DLLs

E

- EDIT 228
 - CHANGE vs. 121
 - exiting 232
 - GO and 295
 - SET CARRY and 484
 - SET REFRESH and 552
- Edit An Expression dialog
 - box 291
- Edit menu, adding to forms 800
- Edit window *See* Table Editor
- EditCopyMenu property 800
- EditCutMenu property 801
- editing 107
 - See also* changing
 - calculated fields 232
 - code 140, 504
 - data 228, 595
 - with BROWSE 105
 - with CHANGE 121
 - expressions 291
 - memo fields 232, 447, 568, 596
 - programs 164
 - restricting 108, 231
 - text files 165, 697
- editing controls 726
- editing tool 682
- editing windows
 - displaying 164, 165
 - setting 568
- editor objects
 - scroll bars 929
 - wordwrapping text 972
- editors, text, alternate 164
 - See also* Text Editor
- EditPasteMenu property 801
- EditUndoMenu property 802
- EJECT 233
 - _peject and 650
- EJECT PAGE 234
 - EJECT vs. 233
 - ON PAGE and 380
- ELAPSED() 235
 - SECONDS() vs. 473
- Element() property 803
 - Subscript() vs. 949
- elements, array *See* array
- elements
 - #else 670
- embedded null characters 330
- empty character strings 330
 - comparing 511
- empty date strings 14, 225
- empty memo fields 330
- EMPTY() 237
 - See also* ISBLANK()
 - BLANK and 102
- Enabled property 804
- encrypting tables 508
- end-of-file indicator 238, 253
- end-of-line characters 256, 275
 - returning 277
- Enter key, simulating Tab 492
- entry fields 700
 - backgrounds 906
 - formatting text 911
 - keystrokes, evaluating 834
 - moving to 489
 - scrolling width 840
 - values, changing 930
- environment commands
 - CHARSET() 123
 - CLEAR 128
 - CREATE SESSION 171
 - DISPLAY MEMORY 211
 - DISPLAY STATUS 212
 - LDRIVER() 327
 - LIST MEMORY 334
 - LIST STATUS 334
 - MEMORY() 356
 - SET 478
 - SET BELL 482
 - SET BORDER 484
 - SET CONFIRM 489
 - SET CONSOLE 489
 - SET DESIGN 502
 - SET DISPLAY 506

- SET EDITOR 506
- SET FULLPATH 518
- SET INTENSITY 526
- SET LDCHECK 529
- SET LDCONVERT 530, 993
- SET MESSAGE 539
- SET ODOMETER 540
- SET SAFETY 558
- SET TALK 562
- SET() 569
- SETTO() 571
- SHELL() 573
- VERSION() 627
- environment variables (DOS) 290
- environments
 - current working 175
 - See also* views; work areas
 - information, getting 212
 - multiuser *See* multiuser environments
 - settings, retrieving 175
- EOF() 238
 - FIND and 260
 - LOCATE and 343
 - RECNO() and 425
 - SEEK and 474
 - SET RELATION and 555
 - SKIP and 579
- equality 21, 117, 267
 - comparing character strings 511
- equal-to operator 21
- ERASE 239
 - DELETE FILE vs. 198
- erasing memo fields 77, 149, 447, 448
 - See also* deleting
- error codes, portability 241, 357
- Error dialog box 140
- error handling
 - See also* debugging commands
 - CERROR() 120
 - DBERROR() 182
 - DBMESSAGE() 184
 - ERROR() 240
 - FERROR() 254
 - LINENO() 333
 - MESSAGE() 357
 - ON ERROR 371
 - ON NETERROR 378
 - ON READERROR 381
 - PROGRAM() 406
 - RETRY 457
 - SET ERROR 509
 - SQLERROR() 587
 - SQLMESSAGE() 589
- error messages 489, 962, 1015–1030
 - customizing 371, 509
 - returning 184, 357
- ERROR() 240
 - resetting 457, 459
- errors
 - compiler, returning 120
 - data entry 381, 962
 - DDE applications 953
 - fixing 140
 - multiuser environments 378
 - resolving 457
 - run-time 371, 378, 509
 - IDAPI 182, 184
 - line numbers, returning 333
 - server 587, 589
 - syntax 140
 - trapping data entry 381
- Esc key 510
 - See also* keyboard; keystrokes
 - disabling 372, 510, 805
- escape sequences 544
- EscExit property 805
- evaluating expressions 615
- evaluating user choices 336, 782
- events 834
 - adding records 857
 - assigning pushbuttons 789, 861
 - closing forms 862
 - conditions 804, 967
 - executing automatically 858, 865
 - keyboard *See* keyboard event commands
 - mouse *See* mouse event commands
 - moving forms 848, 884
 - moving record pointers 885
 - opening forms 887
 - providing help 866
 - resizing forms 899
 - selecting objects 804, 875
 - sizing forms 940
 - trapping
 - DDE applications 863, 886, 889, 890, 901
 - multiuser environments 265, 461
- exact matches (searches) 260, 473
 - failing 539
- exactly equal to operator 21
- exchanging data *See* DDE links
- exclusive mode 513
- exclusive OR operations 101
- Execute() property 806
- executing dBASE commands
 - page formatting 379
 - shortcuts 372, 375, 519, 526
- executing DOS commands 41, 223, 467
- executing events
 - automatically 858, 865
 - closing forms 862
- executing macros, DDE applications 806
- executing programs *See* program execution
- executing SQL statements 588
- exit codes, returning 467
- exiting dBASE 417
- exiting loops 221, 222, 273
- exiting programs 113
- EXP() 241
- exponentiation
 - base e 241
 - exponents, returning 345
 - square roots and 590
- exponentiation operators 20
- exporting tables with COPY 144
- Expression Builder 291
- expression commands
 - GETEXPR() 291
 - MAX() 350
 - MIN() 357
 - TYPE() 615
- expressions 12
 - blank *See* empty
 - changing 667
 - character strings 82, 928
 - comparing 350, 357
 - editing 291
 - empty 316
 - testing for 237, 316
 - evaluating 43, 301, 615
 - fields list 263
 - filters 517
 - finding 82, 346, 927
 - grouping 17
 - identifiers 668
 - key *See* key expressions
 - literal 615
 - logical 350, 357
 - replacing data 439
 - results, viewing 43, 46, 561
 - storing 593
- EXTERN 242
 - BITAND() and 97, 98, 99, 100, 101
 - LOAD DLL and 341

external applications 693, 796,
831, 919
external functions 244, 829

F

FACCESSDATE() 247
FCLOSE() 247
FCREATE() 248
 FCLOSE() and 248
 FEOF() and 253
 FFLUSH() and 255
 FGETS() and 256
 FPUTS() and 275
 FREAD() and 277
 FSEEK() and 278
FCREATEDATE() 250
FCREATETIME() 250
FDATE() 251
FDECIMAL() 252
FEOF() 253
 FGETS() and 257
FERROR() 254
 FGETS() and 257
FFLUSH() 255
FGETS() 256
 FREAD() and 277
field commands
 See also record commands
 APPEND MEMO 77
 BINTYPE() 96
 CLEAR FIELDS 131
 COPY BINARY 145
 COPY MEMO 148
 FDECIMAL() 252
 FIELD() 258
 FLDCOUNT() 262
 FLDLIST() 263
 FLENGTH() 264
 ISBLANK() 316
 MEMLINES() 354
 MLINE() 360
 REPLACE BINARY 443
 REPLACE MEMO 447
 REPLACE
 MEMO...FROM 448
 REPLACE OLE 449
 SET BLOCKSIZE 483
 SET FIELDS 514
 SET MBLOCK 536
 SET MEMOWIDTH 538
 SET WINDOW OF
 MEMO 568
 STORE MEMO 596
field controls *See* entry fields
field descriptor bytes 1006
field names 8, 10

aliases 10, 441, 515
automem variables and 433
browse objects 937
changing 362
display, suppressing 206
displaying 520
getting 258
 passing as parameters 404
field widths, changing 362
FIELD() 258
fields
 accessing specific 515
 adding 363
 asterisks in 440
 binary *See* binary fields
 browse objects 807
 calculated *See* calculated
 fields
 changing data 404, 440
 contents, storing 593
 copying 154, 445
 multiple 144, 150
 counting 262
 definitions, displaying 214
 deleting 363
 empty 316
 filling with blanks 102, 260
 freezing 108, 231
 key *See* key fields
 length 264
 maximums and limits 1002
 memo *See* memo fields
 passing as parameters 404
 sorting on multiple 582
 structure-extended tables 178
 type, returning 615
 updating 442
 variable-length 15
fields list 206
 adding fields 484
 browse objects 807
 clearing 131, 515
 defining 514
 new tables 322, 612
 returning 263
Fields property 807
Fields() property 808
FieldWidth property 809
file attributes
 (DOS) 56, 248, 270, 793
 (Windows95) 794
file buffers, flushing 255
file indicators
 beginning 103
 end 238, 253
file names 8, 162, 414
 changing 437

creating 283
default 143
returning 114, 293
 full paths 518
file pointers 249, 253, 270
files 257
 text files 275, 277, 278, 286
file utilities and information
 ! command 41
 COPY FILE 146
 CREATE FILE 165
 DELETE FILE 198
 DISPLAY FILES 209
 ERASE 239
 FDATE() 251
 FILE() 259
 FSIZE() 280
 FTIME() 281
 FUNIQUE() 283
 GETDIRECTORY() 289
 GETFILE() 292
 LIST FILES 334
 PUTFILE() 414
 RENAME 437
 TYPE 614
FILE() 259
 FCREATE() and 249
 FDATE() and 251
 FSIZE() and 280
 FTIME() and 281
files
 adding to catalogs 486
 backing up 205, 362
 binary 145
 coverage analysis 490
 reading from 443
 closing 135, 247, 346, 417
 copying 146, 626
 coverage 208, 490
 creating 248, 564
 date and time stamps 497,
 563
 returning 251, 281
 deleting 198, 239
 directory listings 203, 204
 end-of-line indicator 256
 file pointer, moving 277, 278
 finding 501, 542
 checking existence 259
 format *See* format files
 header 674
 header structures 1005, 1007
 include 674
 index *See* index files
 information, getting 55, 204,
 793, 794
 file handle numbers 249
 identifier numbers 269

- locks 141, 338
 - size 280
- linking 110
- locking automatically 265, 339, 461
 - disabling locks 532
- low-level 135
- maximums and limits 1003
- memory 453, 469
- menu definition 842
- naming 414
- object 139
- opening 248, 269
- overwriting 558
- position, returning 103, 238
- program *See* program files
- protecting 409
- query 169, 174, 567
- referencing 8
- renaming 437
- saving 255
- selecting 292
- temporary 283, 623
 - SORT and 582
- text *See* text files
- types, supported 73, 302
- unlocking 552, 618
- view 175
- writing to 45, 286
 - streaming output 547
- files list 210, 239, 334
- file-sharing modes 513
- Fill() property 810
- filtering data 117
 - setting filters 517
- filters, queries, and views
 - CREATE QUERY 169
 - CREATE VIEW 174
 - CREATE VIEW...FROM ENVIRONMENT 174
 - DISPLAY 206
 - LIST 334
 - MODIFY QUERY 362
 - MODIFY VIEW 362
 - SET FILTER 516
 - SET VIEW 567
- financial transactions 111
 - future value 284
 - payments 391
 - present value 415
- FIND 259
 - EOF() and 238
 - FOUND() and 274
 - INDEX and 305
 - LOCATE vs. 343
 - SEEK vs. 473, 474
 - SET EXACT and 512
 - SET NEAR and 539
 - SOUNDEX() and 584
- finding data *See* search operations
- First property 811
- FirstIndex property 812
- fixed-length records 143
- FKLABEL() 261
- FKMAX() 262
- FLDCOUNT() 262
- FLDLIST() 263
- FLENGTH() 264
- float data types 14
- float values
 - adding 602
 - averaging 92, 111
 - blank 103
 - comparing 117, 267
 - finding 474, 475
 - key expressions 304
 - length, returning 331
 - replacing 440
 - returning 298, 625
 - absolute values 48
 - angles 51, 84, 90, 91, 157, 226, 577, 608
 - future value 284
 - integer portion 313
 - logarithms 345
 - base e 241
 - pi 395
 - present value 415
 - principal 391
 - sign 576
 - square root 590
 - rounding 464
- FLOCK() 265
 - RLOCK() vs. 461
 - SET REPROCESS and 557
 - UNLOCK and 618
- FLOOR() 267
 - compared 313
- FLUSH 268
 - FTIME() and 281
- FNAMEMAX() 269
- focus 422, 811
 - forms 386
 - getting 757, 865
 - moving 492, 765, 875, 950
 - arrow keys and 822
 - restricting 961
 - setting 934
- FocusBitmap property 813
- Follow property 814
- FontBold property 815
- FontItalic property 815
- FontName property 816
- fonts 44
 - changing 45
 - selecting 294, 816, 925
 - text attributes 44
 - text, objects 815, 818, 819
 - size 817, 926
- fonts commands
 - DEFINE COLOR 194
 - GETFONT() 294
 - ISCOLOR() 317
 - SET COLOR OF 488
 - SET COLOR TO 488
- Fonts dialog box 294
- FontSize property 817
- FontStrikeOut property 818
- FontUnderline property 819
- footers, printing 380
- FOPEN() 269
 - FCLOSE() and 248
 - FEOF() and 253
 - FFLUSH() and 255
 - FGETS() and 256
 - FPUTS() and 275
 - FREAD() and 277
 - FSEEK() and 278
- FOR() 271
- FOR...NEXT 272
 - SLEEP vs. 580
- foregrounds
 - high intensity 778
- form commands
 - CLOSE FORMS 135
 - CREATE
 - APPLICATION 161
 - CREATE FORM 165
 - CREATE MENU 168
 - CREATE POPUP 168
 - CREATE SCREEN 171
 - _curobj 639
 - MODIFY
 - APPLICATION 362
 - MODIFY FORM 362
 - MODIFY MENU 362
 - MODIFY SCREEN 362
 - MSGBOX() 365
 - ON SELECTION FORM 383
 - OPEN FORM 386
 - READMODAL() 421
 - REDEFINE 427
 - SET CUAENTER 492
- Form Designer 161, 165, 171
- Form Expert 166
- form files 166
- form objects
 - See also* objects
 - activating 934
 - aligning 836, 955
 - current 422

- definitions
 - changing 427
 - clearing from memory 435
 - grouping 822
 - linking 785
 - placing 851, 853
 - referencing 811
 - tabbing order 639, 765, 852
 - format files 71
 - BROWSE and 108
 - closing 135
 - EDIT and 231, 232
 - formats
 - character data 612
 - date 487, 612
 - current 116
 - returning 216, 225, 227, 353
 - SLEEP 580
 - specifying 496
 - file, supported 302
 - function templates 44, 820
 - international 496
 - labels 167
 - logical fields 612
 - numeric data 331, 612
 - currency symbols 493
 - decimal separator 545
 - thousands separator 559
 - picture templates 43, 559, 612, 911
 - text 44, 820, 911
 - size 817, 926
 - time 496, 563, 610
 - SLEEP 580
 - formfeeds 233
 - forms 952, 965
 - See also* form commands
 - activating 934
 - adding Edit and Windows menus 719
 - adding menus 168, 716, 719
 - adding popups 168
 - anchoring objects 763
 - borders 836, 954
 - changing 161, 165, 171
 - closing 135, 773, 776, 805, 862
 - Control menu 949
 - control tips 939
 - creating 161, 165, 171, 704
 - default 71
 - designing 791
 - displaying 956, 971
 - formatting data 612
 - maximums and limits 1003
 - MDI 707, 841, 842
 - menu definition files 168, 842
 - modal 386, 421, 918
 - moving 848, 884
 - moving through 903, 929
 - multi-page 751
 - multiple pages 902, 903
 - non-modal 386, 901
 - objects *See* form objects
 - opening 386, 421, 887, 901, 918
 - parent *See* parent forms
 - popup definition files 168
 - pop-up menus 729, 913
 - printing 914
 - scroll bars 929
 - SET DESIGN and 502
 - sizing 764, 839, 843, 899, 971
 - preventing 940
 - specifying text 944
 - submitting 383, 898
 - FOUND() 274
 - CONTINUE and 141
 - FIND and 260
 - LOCATE and 343
 - SEEK and 473
 - SEEK() vs. 476
 - SELECT and 476
 - SET NEAR and 540
 - FPUTS() 275
 - FWRITE() vs. 287
 - FREAD() 277
 - FGETS() and 257
 - FGETS() vs. 257
 - freeing memory 113, 133, 268, 431
 - indexes and 200
 - French date format 496
 - frequency (bell) 482
 - FROM clause 984
 - FSEEK() 278
 - FGETS() and 257
 - FOPEN() and 270
 - FPUTS() and 276
 - FREAD() and 277
 - FSHORTNAME() 279
 - FSIZE() 280
 - FTIME() 281
 - FUNCTION 282
 - function calls 23
 - C conventions 242
 - debugging *See* debugging
 - external 244
 - forms, submitting 898
 - Pascal conventions 242
 - retrying 457
 - function keys
 - assigning
 - character strings 519
 - command execution 375, 519
 - current setting 569, 571
 - default assignments 519
 - name, returning 261
 - number, returning 262
 - programmable 1003
 - function operators 23
 - function pointers 17
 - Function property 820
 - function symbols 612, 820
 - function templates 44, 820
 - functions 4
 - See also* specific dBASE functions
 - abbreviating 7
 - executing 519
 - external 829
 - inline 668, 669
 - key expressions and 304
 - prototypes, DLLs 242, 637
 - referencing 17
 - user-defined *See* UDFs
 - FUNIQUE() 283
 - future value, returning 284
 - FV() 284
 - FWRITE() 286
 - FPUTS() vs. 276
- ## G
-
- GENERATE 288
 - generating random numbers 283, 417
 - German date format 496
 - GETCOLOR() 288
 - DEFINE COLOR and 194
 - GETDIRECTORY() 289
 - GETENV() 290
 - GETEXPR() 291
 - GETFILE() 292
 - GETFONT() 294
 - ? and 44
 - GetTextExtent property 821
 - GO 294
 - BOOKMARK() and 104
 - EOF() and 238
 - implicit GOTO 4
 - SET FILTER and 517
 - SET KEY and 528
 - graphics 145
 - aligning 761
 - centering 762
 - displaying 454, 709
 - printing 455
 - pushbuttons 795, 798, 813, 960

- storing
 - binary fields 146
 - memo fields 149
- greater-than operator 21
- GROUP BY clause 984
- Group property 822
- grouping expressions 17
- grouping objects 822
- Grow() property 823

H

- hard drives *See* disk drives
- hatched (blended)
 - backgrounds 106, 229, 906
- HAVING clause 984
- header files 674
 - changing 674
- header structures (file) 1005, 1007
- Header3D property 824
- headers, printing 380
- headings *See* field names
- Height property 826
 - Alignment vs. 762
 - Width and 969
- HELP 296
- Help system
 - activating 296, 521
 - customizing 866
 - keywords, specifying 828
 - topics, specifying 565, 827
- HelpFile property 827
- HelpID and 828
 - OnHelp and 866
- HelpID property 828
 - HelpFile and 827
 - OnHelp and 866
- hexadecimal numbers
 - decimal equivalents 297
 - returning 320
- hiding cursors 494
- high-intensity attributes 778
- home directory 640
- HOME() 297
- horizontal scroll bars 965
- HTOI() 297
 - FGETS() and 256
 - FPUTS() and 276
 - ITOH() vs. 321
- hWnd property 828

I

- I/O 704
 - delineating 193
 - display

- enabling/disabling 489
- widths, memo fields 538
- environment messages 212, 562
- interrupting 510
- outlining 193
- printing 233
 - page formatting 379
- table structures 214
- I/O commands
 - ? command 43
 - ?? command 46
 - CLOSE ALTERNATE 135
 - CLOSE FORMAT 135
 - CREATE LABEL 167
 - CREATE REPORT 169
 - INPUT 307
 - LABEL FORM 325
 - MODIFY LABEL 362
 - MODIFY REPORT 362
 - REPORT FORM 451
 - SET ALTERNATE 479
 - SET HEADINGS 520
 - SET SPACE 561
 - WAIT 628
- Icon property 829
- icons 829
- ID checking, language
 - drivers 529, 530, 993
- ID property 830
- ID() 298
 - NETWORK() and 369
- IDAPI errors 182, 184
- identifiers
 - See also* names
 - defining 667
 - without replacement text 668
 - multiple programs 674
- objects 830
- replacing with specified values 668
- undefining 669, 676
- IF 299
 - See also* #if
 - DO CASE vs. 219
 - IIF() vs. 301
 - multiple ELSEIFs 299, 300
- #if 670
- #ifdef 671
- #ifndef 672
- IIF() 301
- image objects 708
 - See also* graphics
 - aligning graphics 761
 - displaying data 786
- implicit commands 4

- IMPORT 302
- importing tables 302
- #include 674
- INCLUDE directory 674
- include files 674
 - changing 674
- incompatible data types 363
- incrementing spin box
 - values 946
- _indent 640
- _rmargin and 661
- _wrap and 663
- INDEX 303
 - FIND and 260
 - FOR() and 271
 - REINDEX and 430
 - SEEK and 474
 - SET ESCAPE and 510
 - SET EXCLUSIVE and 513
 - SET INDEX and 525
 - SET UNIQUE and 566
 - SORT vs. 583
 - UNIQUE() and 617
- index call operator 24
- index files
 - See also* .MDX files; .NDX files
 - allocating memory 483, 522
 - closing 135
 - copying 144, 152
 - creating 150, 176
 - deleting 199, 983
 - information, getting 212
 - moving through 295
 - multiple 606
 - names, returning 353, 368, 387, 605
 - opening 524, 622
 - temporary 582
- indexes
 - See also* key expressions; key fields
 - aliases 528
 - copying tables 144
 - creating 271, 303, 323, 979
 - date fields 227, 630
 - deleting 982
 - identical keys 566
 - master 304, 347
 - returning 368
 - names 387, 605
 - specifying 525, 541, 622
 - maximums and limits 1001
 - multiuser environments 200
 - number of active 606
 - numeric fields 598
 - order, reversing 304
 - processing speed 522

- rebuilding 440
- replacing data 440
- SCAN and 471
- sort order, setting 304
- sorting data vs. 583
- SOUNDEX codes 584
- tags 305
 - creating 147, 152
 - deleting 199
 - number, returning 607
 - returning 605
- unique 566, 617
- updating 363, 429, 566
 - APPEND and 71
 - automatically 305
 - EDIT and 231
 - INSERT and 309
- indexing and sorting
 - CLOSE INDEXES 135
 - COPY INDEXES 147
 - COPY TAG 152
 - DELETE TAG 199
 - FOR() 271
 - INDEX 303
 - KEY() 323
 - MDX() 352
 - NDX() 368
 - ORDER() 387
 - REINDEX 429
 - SET IBLOCK 522
 - SET INDEX 524
 - SET KEY TO 527
 - SET ORDER 541
 - SET UNIQUE 566
 - SORT 581
 - TAG() 605
 - TAGCOUNT() 606
 - TAGNO() 607
 - UNIQUE() 617
- indicator, record *See* record
- pointers
- indirect reference 8
- infinity, returning 608
- initializing
 - arrays 60, 596
 - DLLs 340
 - memory variables 129, 341, 399, 412, 593, 624
 - during program
 - suspension 604
 - system 5
- Initiate() property 831
 - Server and 932
- initiation handlers 637
- INKEY() 306, 1009
 - NEXTKEY() and 370
- inline functions 668, 669
- INPUT 307
- input focus *See* focus
- input/output *See* I/O
- INSERT 309
 - SET CARRY and 484
- INSERT AUTOMEM 310
 - SET CARRY and 484
- INSERT BLANK 309
 - INSERT AUTOMEM vs. 311
 - SET CARRY and 484
- INSERT INTO 983
- Insert() property 832
 - Add() vs. 759
 - Grow() vs. 824
- INSPECT() 312
- Inspector
 - opening 312
 - View and 685
- INT() 313
- integers
 - See also* numbers
 - decimal separator 545
 - returning 313, 576
 - equality 117, 267
- interest rates
 - future value 284
 - payments 391
 - present value 415
- international date/time
 - formats 496
- interrupting SLEEP 580, 581
- interrupts, Esc key 510
- invalid data entry 482, 962
- investments *See* financial
- transactions
- involution 241
 - exponents, returning 345
 - square roots and 590
- ISALPHA() 314
- ISBLANK() 316
 - AVERAGE and 93
 - BLANK and 102
- ISCOLOR() 317
- IsIndex() property 833
- ISLOWER() 317
- ISMOUSE() 318
- IsRecordChanged()
 - property 833
- ISTABLE() 319
- ISUPPER() 320
- Italian date format 496
- italic type 44, 815
- ITOH() 320
 - HTOI() vs. 298

J

- Japanese date format 496
- JOIN 321
- joining tables *See* linking and relating
- justification *See* alignment

K

- key codes 376
- key expressions 303, 528
 - linking tables 554
 - matching 260, 473, 540
 - returning 323, 324, 430
- key fields
 - See also* indexes
 - changing 440
 - browse objects 814
 - displaying 108, 231
 - key values
 - duplicate 560
 - restricting 527, 566
 - searching on 475
- Key property 834
- KEY() 323
 - SET INDEX and 525
- KEYBOARD 324
- keyboard
 - accelerators 936
 - default assignments 519
 - programmable keys 1003
- keyboard event commands
 - FKLABEL() 261
 - FKMAX() 262
 - INKEY() 306
 - NEXTKEY() 369
 - ON ESCAPE 372
 - ON KEY 375
 - SET CURSOR 494
 - SET ESCAPE 510
 - SET FUNCTION 519
 - SET KEY 526
- Keyboard() property 835
- KeyHigh 809
- KEYMATCH() 324
- keystrokes
 - assigning
 - command execution 372, 375, 519, 526
 - interrupts 510
 - evaluating 834, 859, 868, 869
 - simulating 492, 835
 - values, returning 306, 369
- keywords 4
 - abbreviating 7
 - SET() and 569

alternatives 29
scope 11

L

- label files 167
- LABEL FORM 325
- labels
 - See also* field names; file names
 - displaying 326
 - formatting 167, 612
 - objects 952
 - printing 326
 - SET DESIGN and 502
- landscape orientation 124
- language drivers 123, 327
 - current, returning 327
 - ID checking 529, 530, 993
 - ISALPHA() and 315
 - ISLOWER() and 317
 - ISUPPER() and 320
 - LIKE() and 332
 - LOWER() and 348
 - primary/secondary weights 511
 - PROPER() and 408
 - SOUNDEX() and 584
 - UPPER() and 621
- language elements 3
 - backward compatibility 989
 - unsupported 998
- large numbers 440, 613
- LASTKEY() 327
- LDRIVER() 327
- leading spaces 260, 331
 - deleting 349
- Left property 836
 - Bottom and 772
 - Right and 924
 - Top and 955
- LEFT() 328
- LEN() 330
 - LEFT() and 328
 - SUBSTR() and 601
- LENNUM() 331
 - LEN() vs. 330
- less-than operator 21
- LIKE() 331
 - DIFFERENCE() vs. 203
 - SOUNDEX() and 584
- line lengths, memo fields 354, 360
- line objects 710, 772, 924
 - patterns 908
- linear control structures 219, 299, 301
- linefeeds 233
 - automatic 234
 - character, counting 328, 330, 460
 - substrings 89, 601
 - files 256
- LineNo property 837
- LINENO() 333
 - PROGRAM() and 407
- LinkFileName property 837
- linking and relating
 - JOIN 321
 - RELATION() 430
 - SET RELATION 553
 - SET SKIP 560, 619
 - TARGET() 609
 - UPDATE() 619
- linking files 110
- links
 - DDE 831, 932, 955
 - disabling 951
 - form objects to tables 785
 - OLE 449, 837
 - setting 554
- LIST 334
 - DISPLAY vs. 206, 207
 - EOF() and 238
 - SET HEADINGS and 520
 - SET PRINTER and 547
 - TRANSFORM() and 613
- list boxes 712
 - displaying data 786
 - multiple choices 849
 - prompts 782, 942
 - current 784, 931
 - returning 335, 337
 - selecting 714, 897
- LIST COVERAGE 334
 - DISPLAY COVERAGE vs. 209
- LIST FILES 334
 - DISPLAY FILES vs. 210
 - SET DATABASE and 210
 - SET DBTYPE and 210
 - SET SEPARATOR and 559
- LIST MEMORY 334
 - DISPLAY MEMORY vs. 212
 - LOCAL and 341
 - PRIVATE and 399
- LIST STATUS 334
 - DISPLAY STATUS vs. 213
- LIST STRUCTURE 334
 - DISPLAY STRUCTURE vs. 215
 - RECSIZE() and 426
- LISTCOUNT() 335
 - LISTSELECTED() and 337
- LISTSELECTED() 337
 - DataSource and 787
 - Multiple and 849
- literal array object 680
- literal character strings 13
- literal characters 332
- literal statements 17
- literal values 9
 - dates 14
- literals, expressions 615
- LKSYS() 338
 - CONVERT and 142
- _lmargin 642
 - _alignment and 635
 - _ploffset and 656
 - _rmargin and 661
 - _wrap and 663
- LOAD DLL 340
 - RELEASE DLL and 435
- LOCAL 341
 - PRIVATE vs. 400
 - procedures and 401
- local SQL commands 977–986
- local variables
 - See also* memory variables
 - declaring 341
 - as static 591
 - initializing 341
 - scope, resetting 459
- LOCATE 342
 - CONTINUE and 140
 - EOF() and 238
 - FIND vs. 260
 - FOUND() and 274
 - SEEK vs. 473, 474
 - SET KEY and 528
 - SOUNDEX() and 584
- locating data *See* search operations
- LOCK() 344
 - RLOCK() vs. 462
 - SET REPROCESS and 557
 - UNLOCK and 618
- locks
 - record 344, 461
 - information, getting 141, 338
 - releasing 552, 618
 - retry messages 557
 - table 265, 532
 - information, getting 141, 338
 - releasing 552, 618
- LOG() 344
 - EXP() vs. 241, 345
- LOG10() 345
- logarithms 345

- base e 241
 - logical data types 14
 - logical expressions
 - comparing 350, 357
 - returning values 301
 - logical fields 14
 - blank values 103
 - converting to character 363
 - converting to numeric 363
 - formatting 612
 - text files 144
 - logical operators 21
 - LOGOUT 346
 - LOOKUP() 346
 - EOF() and 238
 - FOUND() and 274
 - INDEX and 305
 - loops 220, 222, 272, 336, 337, 470, 782
 - losing data 255, 362, 389, 440
 - browse objects 814
 - minimizing loss 481
 - losing text 597
 - LOWER() 348
 - ASCAN() and 82
 - AT() and 89
 - LIKE() and 332
 - RAT() and 420
 - Scan() and 928
 - lowercase letters 348
 - converting to uppercase 304, 620
 - first letter 408
 - sorting data 582
 - testing 317
 - low-level access commands
 - FCLOSE() 247
 - FCREATE() 248
 - FEOF() 253
 - FERROR() 254
 - FFLUSH() 255
 - FGETS() 256
 - FOPEN() 269
 - FPUTS() 275
 - FREAD() 277
 - FSEEK() 278
 - FWRITE() 286
 - low-level files 135
 - LTRIM() 349
 - TRIM() vs. 613
 - LUPDATE() 350
- M**
-
- macros
 - executing, DDE applications 806
 - macro expansion
 - (defined) 669
 - nesting 669
 - magnitude (defined) 48
 - manipulating data 117, 581
 - See also* searching and summarizing; sorting data
 - manipulating dates 179, 225
 - margins, setting 534
 - marking records for
 - deletion 196, 200
 - preventing 108, 231, 789, 936
 - removing marks 423
 - master index 304, 347
 - See also* .MDX files
 - returning 368
 - names 387, 605
 - specifying 525, 541, 622
 - master procedures (defined) 458
 - MAX() 350
 - Maximize buttons, enabling 839
 - Maximize property 839
 - maximum values, returning 111
 - MaxLength property 840
 - MCOL() 351
 - MD 352
 - MDI (Multiple Document Interface) 841
 - MDI forms 573, 707, 841, 842
 - MDI property 841
 - Maximize and 839
 - Minimize and 844
 - Moveable and 848
 - Sizeable and 940
 - SysMenu and 949
 - MDOWN() 352
 - .MDX files
 - allocating memory 522
 - converting to tags 152
 - copying 144, 152
 - creating 150, 176
 - deleting tags 199
 - name, returning 352
 - MDX() 352
 - MDY() 353
 - member call operator 24
 - members *See* object classes
 - MEMLINES() 354
 - LEN() vs. 330
 - STORE MEMO and 597
 - memo data types 15
 - memo fields 15, 147, 362, 437
 - allocating memory 483, 536
 - case
 - converting 348, 620
 - first letter only 408
 - testing 314, 317, 320
 - changing 440, 447
 - combining records 322
 - contents, storing 594
 - converting to character fields 440
 - copying 76, 144, 148
 - text files to 77, 448
 - to text files 144
 - creating 483, 537
 - deleting
 - spaces 349, 466, 613
 - specific characters 599
 - display width, setting 538
 - displaying 232
 - editing 232, 447, 568, 596
 - empty 330
 - line lengths 354, 360
 - number of characters 328, 330, 460
 - number of lines 354
 - overwriting 77, 149, 447, 448
 - phonetic values 203, 584
 - repeating 450
 - special effects *See* binary fields
 - substrings, finding 89, 419, 601
 - text
 - See also* text centering 118
 - losing 597
 - returning 360
 - storing 596
 - types 597
 - memo files
 - deleting 983
 - opening 621
 - memory
 - allocating 268
 - indexes 483, 522
 - memo fields 483, 536
 - checking available 356
 - clearing
 - object definitions 921
 - unallocated 268
 - freeing 113, 133, 268, 431
 - indexes and 200
 - managing 133
 - running out of 604
 - memory blocks 537
 - size, changing 522
 - memory files, creating 453, 469
 - memory variables 4, 5, 307, 441
 - arrays 187, 596
 - assigning to expressions 291
 - automem *See* automem variables
 - clearing 131, 432, 453

- non-public 458
- program execution and 113, 417
- copying 453
- current settings, program 569, 571
- decrementing/incrementing 273
- defined 10
- deleting 431
- evaluating 615
- information, getting 211
- initializing 129, 341, 399, 412, 593, 624
 - during program suspension 604
- local *See* local variables
- maximums and limits 1003
- objects 190, 427
- overriding 668
- passing as parameters 403
- prefixes 10, 594
- preserving 412, 453
- private *See* private variables
- public 412, 453
- releasing *See* clearing
- resetting 129
- saving 453, 469
- scope 10, 341, 399, 412, 591, 593
 - resetting 459
- sessions and 593
- static 459, 591
- storing 679
- storing data 595
- storing expressions 593
- storing values 92, 112
- substituting 977
- testing 272
- values
 - finding 474
 - retaining in memory 591
- memory variables commands
 - ACOPY() 49
 - ADEL() 52
 - ADIR() 55
 - AELEMENT() 57
 - AFIELDS() 59
 - AFILL() 60
 - AGROW() 61
 - AINS() 64
 - ALEN() 67
 - ARESIZE() 78
 - ASCAN() 82
 - ASORT() 84
 - ASUBSCRIPT() 87
 - CLEAR MEMORY 131
 - DECLARE 187
 - LOCAL 341
 - PRIVATE 399
 - PUBLIC 412
 - RELEASE 431
 - RESTORE 453
 - SAVE 469
 - STATIC 591
 - STORE 593
- MEMORY() 356
- memory-intensive tasks 133
- Menu Builder 842
- menu commands
 - choosing 936
 - shortcuts 943
- menu definition files 168, 842
- Menu Designer 168
- menu objects 716
- MENU() 356
- MenuFile property 842
- menus
 - See also* pop-up menus
 - adding checkmarks 774
 - generating 168, 716, 719, 842
 - initializing 867
 - separators 931
- MESSAGE() 357
- messages
 - See also* error messages;
 - prompts
 - confirmation 558
 - displaying 489, 562, 580
 - current environment 212
 - in status bars 945, 962
 - memory variables 211
 - environment
 - information 212, 562
 - file locking 557
 - invalid data entry 962
 - status bar 539
- methods 5
 - See also* procedures
 - object classes 126
- MIN() 357
- Minimize buttons, enabling 843
- Minimize property 843
- minimum values, returning 111
- MKDIR 359
 - MD vs. 352
- MLINE() 360
 - LEN() and 330
 - LTRIM() and 349
- MOD() 360
- modal forms 386, 421, 918, 956
- Mode property 844
 - Toggle and 954
- modeless windows *See* non-modal forms
- MODIFY 361
- MODIFY... commands
 - SET DESIGN and 502
- MODIFY APPLICATION 362
- MODIFY COMMAND
 - CREATE FILE vs. 165
 - SET DEVELOPMENT and 503
- MODIFY FORM 362
- MODIFY LABEL 362
 - CREATE LABEL and 167
 - LABEL FORM and 326
- MODIFY MENU 362
- Modify property 845
- MODIFY QUERY 362
- MODIFY REPORT 362
 - REPORT FORM and 452
- MODIFY SCREEN 362
- MODIFY STRUCTURE 362
 - CREATE...FROM vs. 176
- MODIFY VIEW 362
- modifying table structures 978
- modulus, returning 360
- monetary values 493
- monitors
 - See also* screens
 - color attributes 778
 - display modes 506
 - intensity, setting 778
- monochrome monitors
 - intensity, setting 778
- MONTH() 364
- mouse buttons
 - clicking 871
 - middle button 876, 878, 893
 - twice 869, 876, 891
 - releasing 873, 880, 895
- mouse drivers 318
- mouse event commands
 - INKEY() 306
 - ISMOUSE() 318
 - ON MOUSE 378
- mouse events
 - assigning 871, 873, 878, 880, 893, 895
 - double-clicks 870, 876, 891
 - moving forms 848
 - sizing forms 940
- mouse pointer
 - changing 846, 882
 - moving 846, 882
- MousePointer property 846
- MOVE WINDOW 365
- Move() property 847
- Moveable property 848
- moving

- file pointers 257, 275, 277, 278, 286
- forms 848, 884
- objects 847
- record pointers 294, 471, 481, 578
 - linked tables 560
- moving between records 109
 - Table Editor 232
- moving through forms 903, 929
- moving through tables 294, 560
- MROW() 365
- MSGBOX() 365
- multi-dimensional arrays *See* arrays
- multi-line comments 42
- multi-page forms 751
- multiple conditions, testing 300
- Multiple Document Interface (MDI) 841
- multiple documents, opening 707, 841
- multiple ELSEIF statements 299, 300
- multiple fields
 - changing data 440
 - copying 144, 150
 - sorting data 582
- multiple forms, opening 386
- multiple index files, returning 606
- multiple programs
 - compiling 674
 - identifiers 674
- Multiple property 849
- multiple-choice list boxes 336, 337, 782, 849
- multiplication operator 20
- multi-tables, comparing records 322
- multiuser environments
 - See also* shared data
 - commands
 - changing data 122, 265, 344, 461
 - counting records 159
 - deleting records 630
 - errors 378
 - file-sharing modes 513
 - indexes 200
 - releasing locks 552, 618
 - saving data 138
 - screens, refreshing 552
 - setting locks 265, 344, 461, 532
 - retry messages 557
 - testing for 369

- transactions 94
 - committing 138
 - rolling back 463
- undoing changes 463
- updating data 138, 265, 461
- user names, returning 298

N

- Name property 850
- names 8
 - See also* aliases; identifiers
 - catalog files 114
 - databases 180
 - DDE applications, returning 932, 955
 - directory 359
 - field *See* field names
 - file *See* file names
 - index files, returning 353, 368, 387, 605
 - objects, returning 850
 - procedures 401
 - table
 - changing 438
 - default 144
 - returning 183, 609
 - work areas 69
- natural logarithms 345
 - base e 241
- .NDX files
 - converting to tags 147
 - copying 152
 - name, returning 368
 - specifying as master 622
- NDX() 368
- SET INDEX and 525
- negative values
 - absolute 48
 - finding 576
- nested DOs 217
- nesting
 - character strings 13
 - DO CASE structures 220
 - DO WHILE loops 221
 - DO...UNTIL loops 222
 - FOR...NEXT loops 274
 - macros 669
 - SCAN loops 471
- net present values 111
- network drives 626
- NETWORK() 369
- networks *See* multiuser environments; shared data
- commands
- NEW operator 19, 22, 741
- REDEFINE and 427
- new tables, creating 178
- NextCol() property
 - NextRow() and 853
- NextCol() property 851
- NextIndex() property 852
- NEXTKEY() 369
 - INKEY() and 306
- NextObj property 852
- NextRow() property 853
 - NextCol() and 851
- non-dBASE formats 302
- non-modal forms 386, 901
- not equal to operator 21
- NOTE 42
 - && vs. 42
- Notify() property 854
- null characters 330
- numbers
 - See also* decimal digits; integers
 - adding 111, 602, 611
 - averaging 92, 111
 - constants 668
 - dividing 360
 - hexadecimal
 - decimal equivalents 297
 - returning 320
 - incrementing 946
 - large 440, 613
 - negative 576
 - absolute values 48
 - random 283, 417
 - rounding 464, 598
 - sign, determining 576
- numeric data
 - bitwise operations 97, 99
 - exclusive OR 101
 - return values, getting 100
 - shift bits 98, 99
- comparing 350, 358
 - equality 117, 267
- converting characters 363
- converting to character 598
- converting to strings 598
- finding 474, 475
- formatting 331, 612
 - currency symbols 493
 - decimal separator 545
 - thousands separator 559
- key expressions 304
- precision, setting 546
- replacing 440
- returning
 - absolute values 48
 - characters as 363, 625
 - integer portion 313
 - logical 363

- truncating 313
- numeric data commands
 - ABS() 48
 - ACOS() 50
 - ASIN() 84
 - ATAN() 90
 - ATN2() 91
 - CEILING() 117
 - COS() 157
 - DTOR() 226
 - EXP() 241
 - FLOOR() 267
 - FV() 284
 - INT() 313
 - LENNUM() 331
 - LOG() 344
 - LOG10() 345
 - MOD() 360
 - PAYMENT() 391
 - PI() 394
 - PV() 415
 - RANDOM() 417
 - ROUND() 464
 - RTOD() 465
 - SET CURRENCY 493
 - SET DECIMALS 500
 - SET POINT 545
 - SET PRECISION 546
 - SET SEPARATOR 559
 - SIGN() 576
 - SIN() 577
 - SQRT() 590
 - TAN() 608
- numeric data types 14
- numeric fields
 - blank values 103
 - changing 440
 - indexing 598
 - length, returning 331
 - structure-extended tables 178
 - text files 144
- numeric operators 20

O

- object classes 5–6, 189, 775
 - CLASS ARRAY 679
 - CLASS ASSOCARRAY 681
 - CLASS BROWSE 682
 - CLASS CHECKBOX 686
 - CLASS COMBOBOX 690
 - CLASS DDELINK 693
 - CLASS DDETOPIC 694
 - CLASS EDITOR 697
 - CLASS ENTRYFIELD 700
 - CLASS FORM 704
 - CLASS IMAGE 708
 - CLASS LINE 710
 - CLASS LISTBOX 712
 - CLASS MENU 716
 - CLASS MENUBAR 719
 - CLASS OBJECT 721
 - CLASS OLE 721
 - CLASS
 - OLEAUTOCLIENT 725
 - CLASS PAINTBOX 726
 - CLASS POPUP 729
 - CLASS PUSHBUTTON 731
 - CLASS RADIOBUTTON 735
 - CLASS RECTANGLE 739
 - CLASS SCROLLBAR 742
 - CLASS SHAPE 745
 - CLASS SPINBOX 746
 - CLASS TABBOX 749
 - CLASS TEXT 752
 - creating 126, 721
 - creating new 6
 - declaring 125
 - derived 6
- object files 139
- object handles 829
- object operators 22
- object pointers 811
- object reference variables 19
 - NEW operator 22
- object-reference data types 19
- objects 636
 - adding borders 770, 771
 - aligning 836, 954
 - anchoring 763
 - attributes *See* properties
 - browse 824
 - clearing from memory 921
 - CLEAR MEMORY
 - and 131
 - colors, setting 777
 - creating 5, 22, 189, 726
 - current 422
 - finding 757
 - default 788
 - definition
 - changing 427
 - clearing from memory 435
 - dimensions, setting 826, 969
 - displaying data 786, 807, 844, 953
 - editing tool 682
 - focus 811, 934
 - getting 757, 865
 - moving 492, 765, 875, 950
 - arrow keys and 822
 - restricting 961
 - fonts 815, 818, 819
 - selecting 816, 925
 - grouping 822
 - labeling 952
 - linking 785
 - moving 847
 - names, returning 850
 - overlapping, preventing 851
 - parent form 905
 - placing in forms 851, 853, 903
 - read-only 787
 - referencing 19, 830
 - forms 811
 - selecting 804, 968
 - See also* focus, moving
 - sizing 847
 - tabbing order 639, 765, 852
 - text
 - formatting 820, 911
 - size, setting 817, 926
 - values, changing 930
 - viewing 967
- objects commands
 - _app 636
 - CLASS...ENDCLASS 125
 - DEFINE 189
 - INSPECT() 312
 - LISTCOUNT() 335
 - LISTSELECTED() 337
 - PLAY SOUND 395
 - RELEASE OBJECT 435
 - RESTORE IMAGE 454
 - SHOW OBJECT 575
- OEM conversions 69
- OEM()
 - ANSI() and 69
- OldStyle property 855
- OLE data types 15
- OLE documents
 - adding 449
 - defined 838
- OLE fields
 - adding OLE documents 449
 - information, getting 855
 - writing to 449
- OLE files 837
- OLE links 449, 837
- OLE objects 722
- OLE server applications 721, 725
 - accessing 796, 933
- OleType property 855
- ON BAR 371
- ON ERROR 371
 - LINENO() and 333
 - ON ESCAPE vs. 373
 - ON NETERROR vs. 378
 - PROGRAM() and 407
 - RETRY and 457
 - SET ERROR vs. 509
- ON ESCAPE 372
 - ON ERROR vs. 371

- ON KEY vs. 375
 - ON EXIT BAR 374
 - ON EXIT MENU 374
 - ON EXIT PAD 374
 - ON EXIT POPUP 374
 - ON KEY 375
 - FKLABEL() and 261
 - ON ERROR vs. 371
 - ON ESCAPE vs. 373
 - SET KEY and 526
 - ON MENU 377
 - ON MOUSE 378
 - ON NETEROR 378
 - ON PAD 379
 - ON PAGE 379
 - EJECT PAGE and 234
 - ON POPUP 381
 - ON READERROR 381
 - ON SELECTION BAR 382
 - ON SELECTION FORM 383
 - ID property and 830
 - OnSelection vs. 898
 - ON SELECTION MENU 384
 - ON SELECTION PAD 384
 - ON SELECTION POPUP 384
 - OnAdvise property 856
 - OnAppend property 857
 - OnChange property 858
 - OnChar property 859
 - OnClick property 861
 - ShortCut and 936
 - OnClose property 862
 - one-dimensional arrays *See* arrays
 - one-to-many relationships 560
 - OnExecute property 863
 - OnFormSize property 864
 - OnGotFocus property 865
 - OnSize and 900
 - OnHelp property 866
 - HelpFile and 827
 - HelpID and 828
 - OnInitiate 636
 - OnInitMenu property 867
 - OnKeyDown property 868
 - OnKeyUp property 869
 - OnLeftDbClick property 869
 - OnLeftMouseDown
 - property 871
 - OnLeftMouseUp property 873
 - online Help *See* Help system
 - OnLostFocus property 875
 - OnMiddleDbClick
 - property 876
 - OnMiddleMouseDown
 - property 878
 - OnMiddleMouseUp
 - property 880
 - OnMouseMove property 882
 - OnMove property 884
 - OnNavigate property 885
 - OnNewValue property 886
 - Advise() and 760
 - OnOpen property 887
 - OnPaint property 888
 - OnPeek property 889
 - OnPoke property 890
 - OnRightDbClick property 891
 - OnRightMouseDown
 - property 893
 - OnRightMouseUp property 895
 - OnSelChange property 897
 - OnSelection property 898
 - ID property and 830
 - OnSize property 899
 - OnUnadvise property 901
 - OPEN DATABASE 385
 - SET DATABASE and 495
 - Open File dialog box 9
 - OPEN FORM 386
 - Close and 776
 - Open() and 902
 - Open() property 901
 - opening
 - catalog files 485
 - databases 385
 - Debugger 185, 506, 562
 - Expression Builder 291
 - files 248, 269
 - Form Designer 161, 165, 171
 - forms 386, 421, 887, 901, 918
 - index files 524, 622
 - Inspector 312
 - memo files 621
 - Menu Designer 168
 - program files 531, 549
 - query files 567
 - Report Designer 167, 169
 - Table Designer 161, 362
 - tables 9, 478, 621
 - default, setting 499
 - file-sharing modes 513
 - operands (defined) 20
 - operating system 388
 - See also* DOS; system
 - operators 19–26
 - assignment 20
 - binary 20
 - bitwise 100
 - AND 97
 - OR 99
 - shift bits 98, 99
 - XOR 101
 - comparison 511
 - exponentiation 20
 - function 23
 - logical 21
 - numeric 20
 - object 22
 - precedence 26
 - relational 20
 - string 22
 - unary 20
 - optimizing
 - data processing 481
 - memory allocation 268, 537
 - program execution 133, 504
 - search operations 260, 347, 474
 - source code 669
 - option buttons *See* radio buttons
 - options
 - command 4, 7, 28
 - compiler, setting 675
 - OR bitwise operator 99
 - ORDER BY clause 984
 - ORDER() 387
 - organizing data 304, 583
 - orientation, print, setting 124
 - OS() 388
 - outlining output 193
 - output devices 398
 - output *See* I/O; streaming output
 - overlapping objects,
 - preventing 851
 - overstriking text 44, 45, 818
 - overwriting data 440
 - binary fields 145
 - confirmation messages 558
 - memo fields 77, 149, 447, 448
- ## P
-
- PACK 389
 - DELETE vs. 196
 - RECALL and 423
 - SET ESCAPE and 510
 - PAD() 390
 - padding characters 119
 - padding strings 598
 - PADPROMPT() 390
 - _padvance 643
 - page orientation 124
 - PageCount() property 902
 - page-handling routines 234
 - _pageno 644
 - PageNo property 903
 - paging through text 929
 - paintbox objects
 - redrawing 864, 888

- paper size, setting 124
- Paradox data types 980
- Paradox tables 15, 144
 - copying 152
 - creating 161, 176, 322
 - deleting 199
 - deleting records 196
 - indexing 199, 303, 528, 541
 - key expressions, finding 324
 - primary indexes 200, 303
 - secondary indexes 200, 622
- linking 554
- moving to specific records 104
- opening 622
- querying 588
- renaming 438
- sorting data 582
- structures
 - changing 362
 - copying 150, 155
 - designing 177
- PARAMETERS 391
 - parameters 401–406
 - See also* arguments
 - finding number of 394
 - passing 185, 401, 403
 - arrays as 404
 - DLL function
 - prototypes 243
 - fields as 404
 - memory variables as 403
 - properties as 403
 - returning information on 394
- parent forms 905
 - aligning objects 954
 - moving objects 847
 - opening 887
 - sizing objects 847
- Parent property 905
- parent tables 554
 - moving through 560
- Pascal calling conventions 242
- passing arrays as parameters 404
- passing fields as parameters 404
- passing memory variables as parameters 403
- passing properties as parameters 403
- Paste() property 906
- pasting text 801, 906
- paths, directory *See* directory paths
- pattern matching 332
- PatternStyle property 906
- pausing program execution *See* suspending program execution
- PAYMENT() 391
- payments
 - future value 284
 - present value 415
 - principal balance 391
- _pbpage 645
- _pepage and 651
- PCOL() 393
 - _pcolno and 647
 - SET PCOL and 543
- _pcolno 646
- _pcopies 647
- PCOUNT() 394
- _pdriver 648
- Peek() property 907
- _peject 649
- Pen property 908
- PenStyle property 909
- PenWidth property 910
- _pepage 650
- _pbpage and 645
- performance *See* optimizing; processing speed
- _pform 651
- phonetic matches 202, 584
- PI() 394
 - ACOS() and 51
- Picture property 911
- picture templates 43, 559, 612, 911
- placing objects in forms 851, 853
- PLAY SOUND 395
- _plength 653
 - _padvance and 643
 - _porientation and 657
- _plineno 654
- _plength and 644
- _ploffset 656
- POINT parameter 545
- pointers
 - alias 555
 - file 249, 253, 270
 - moving 257, 275, 277, 278, 286
 - function 17
 - object 811
 - record 309
 - linked tables 560
 - moving 294, 471, 481, 578
 - events and 885
 - position, returning 103, 238
 - work areas 476
- Poke() property 912
- population statistics 111
- pop-up controls *See* list boxes
- popup definition files 168
- pop-up menus 729, 958
 - See also* menus
 - initializing 867
- POPUP() 397
- PopupMenu property 913
- popups, generating 168
- _porientation 656
- portrait orientation 124
- positive values, finding 576
- _ppitch 657
 - _pcolno and 647
 - _rmargin and 661
 - _tabs and 662
- _pquality 658
- #pragma 675
 - COVERAGE and 490
- precedence 26
- predefined table structures 162
- prefixes, memory variables 10, 594
- preprocessor
 - See also* compiling
 - call chain 674
 - macros, nesting 669
 - search-and-replace operations 669, 676
- preprocessor directives 667–676
 - defined 5
- present value, returning 415
- preserving memory
 - variables 412, 453
- primary keys *See* key fields
- principal 391
 - future value 284
 - present value 415
- Print() property 914
- printer control codes 544
- printer drivers 1003
- Printer Setup dialog box 123
- printers
 - escape sequences 544
 - horizontal printing
 - position 393, 543
 - specifying 123, 547
 - vertical printing position 411, 551
- printing
 - boxes 193
 - conventions, documentation 2
 - syntax 7
 - data 45, 206, 397
 - advancing paper 233, 234
 - page formatting 379

- print options 123
 - setting margins 534
- environment
 - information 212, 562
- files list 210
- forms 914
- graphics 455
- headers and footers 380
- labels 326
- reports 233, 452
- text files 614
- printing commands
 - See also* dBASE IV printing commands
 - CHOOSEPRINTER() 233
 - DEFINE BOX 193
 - EJECT 233
 - EJECT PAGE 234
 - ON PAGE 379
 - PCOL() 393
 - PRINTJOB 397
 - PROW() 411
 - SET MARGIN 534
 - SET PCOL 543
 - SET PRINTER 547
 - SET PROW 551
- printing data 206
- PRINTJOB
 - _pcopies and 648
 - _peject and 649
- PRINTJOB...ENDPRINTJOB 397
- PRINTSTATUS() 398
- PRIVATE 399
 - LOCAL vs. 341
 - procedures and 402
- private variables
 - See also* memory variables
 - clearing 113, 432, 453
 - declaring 399
 - initializing 399
 - scope, resetting 459
- PROCEDURE 401
- procedure calls 23, 401
 - call chain 218
 - debugging *See* debugging
 - forms, submitting 898
 - recursive 217
 - retrying 457
- procedures 401–406
 - accessing 531
 - closing 135
 - compiling automatically 503
 - coverage analysis 208, 490
 - debugging 185, 186, 406
 - declaring 401
 - execution, stopping 458
 - master, defined 458
 - maximums and limits 1002
 - naming 401
 - referencing 17
 - returning to 458
 - running 217, 458, 549
 - shortcuts 526
 - processing data 11, 375
 - optimizing 481
 - specific records 516, 527
 - specified ranges 526
 - processing speed 504, 562
 - CLEAR PROGRAM and 133
 - FLUSH and 268
 - indexes 522
 - returning 235
 - program calls 218
 - recursive 371, 378
 - program commands
 - See also* Windows programming commands
 - && 41
 - * 42
 - BUILD 110
 - CANCEL 113
 - CLEAR PROGRAM 133
 - COMPILE 139
 - CREATE COMMAND 164
 - DO 217
 - DO CASE 219
 - DO WHILE 220
 - DO...UNTIL 222
 - FOR...NEXT 272
 - FUNCTION 282
 - IF 299
 - IIF() 301
 - PCOUNT() 394
 - PROCEDURE 401
 - QUIT 417
 - RETURN 458
 - SCAN 470
 - SET DEVELOPMENT 503
 - SET LIBRARY 531
 - SET PROCEDURE 549
 - SLEEP 580
 - program execution 139, 217, 458
 - benchmarks 235, 472
 - canceled 113
 - conditional 219, 299, 301
 - OS() 388
 - VERS() 627
 - coverage analysis 675
 - delaying 580
 - interrupting 510
 - optimizing 133, 504
 - problems with 490
 - repeating 220
 - resuming 456, 458, 604
 - retrying 457
 - shortcuts 526
 - stopping 113, 417, 458, 603
 - submitting forms 383
 - suspending 113, 603
 - for specified duration 580
 - viewing 186
- program files 139
 - catalogs and 163
 - closing 113, 549
 - creating 164
 - opening 531, 549
 - procedures and 404
 - recompiling 503
- PROGRAM() 406
 - LINENO() and 333
- programmable function
 - keys 1003
- programs
 - calling *See* program calls
 - changing suspended 604
 - clearing from memory 113, 133
 - compiling *See* compiling
 - coverage analysis 208, 490
 - creating 506
 - current settings 569, 571
 - debugging *See* debugging
 - developing 490, 504
 - documenting 41, 42
 - editing 164
 - executing *See* program execution
 - exiting 113
 - flow, tracking 333
 - interrupting SLEEP 580, 581
 - multiple, identifiers 674
 - names, returning 406
 - recompiling 674
 - running *See* program execution
 - stepping through 186
 - testing 120, 288, 490
 - version control 670, 674
 - viewing 45
- progress bars *See* record counters
- PROMPT() 407
- prompts 489
 - combo boxes 691, 787
 - list boxes 335, 337, 714, 782
 - current 784, 931
 - displaying 787
 - selecting 897
 - tab boxes, current 784
- PROPER() 408
 - ASCAN() and 82
 - Scan() and 928
- properties

See also specific dBASE
property
changing 312
custom classes 126
defined 5, 190, 427
passing as parameters 403
viewing 312

property names 28
property sheets *See* Object
Inspector

PROTECT 409

protecting code 140

protecting data 409, 502

protecting files 409

prototypes

defined 244

DLL functions 242, 637

PROW() 411

_plinen and 654

SET PROW and 551

pseudo-functions *See* inline
functions

_pspacing 659

PUBLIC 412

public variables

See also memory variables

clearing 453

declaring 412

pushbuttons 731, 943

adding graphics 795, 798,
813, 960

assigning actions 789, 861

default 788

disabling 795

submitting forms 898

PUTFILE() 414

PV() 415

Q

.QBE files 169, 174

linking tables 554

names, returning 965

opening 567

qualified field names 10

Quattro Pro 760, 796, 831, 854,
886, 889, 907, 912, 919, 932, 958

queries

See also filters, queries, and
views

DDE applications 907

SET DESIGN and 502

SQL 977

tables, accessing 761

Query Designer 169, 174

question mark (?)

temporary files 283

wildcard character 332
directory listings 9, 204,
210
fields list 516

QUIT 417

quitting dBASE 417

quitting loops 221, 222, 273

R

radians

arccosine 51

arcsine 84

arctangent 90, 91

converting degrees 465

cosine 157

returning 226

sine 577

tangent 608

radio buttons 735

displaying 855

random access memory

(RAM) 356

random numbers 283, 417

random records 288

RANDOM() 417

RangeMax property 915

RangeMin and 916

RangeRequired and 917

Valid vs. 961

RangeMin property 916

RangeMax and 915

RangeRequired and 917

Valid vs. 961

RangeRequired property 917

ranges

key fields 527

out-of-bounds 381

spin boxes 915, 916, 917

RAT() 419

AT() and 89

READ 421

VARREAD() and 627

READKEY() 421, 1009

ReadModal() property 918

READMODAL() vs. 422

READMODAL() 421

MDI and 842

ReadModal() and 918

read-only access 516, 532

objects 787

RECALL 423

RECCOUNT() 424

COUNT vs. 160

DISKSPACE() and 205

RECNO() 425

GO vs. 295

SET RELATION and 554
recompiling programs 503, 674
Reconnect() property 919
record commands

See also field commands

APPEND 70

APPEND AUTOMEM 72

APPEND BLANK 71

APPEND FROM ARRAY 75

BLANK 102

BOF() 103

BOOKMARK() 104

BROWSE 105

CHANGE 121

CLEAR AUTOMEM 129

COPY TO ARRAY 153

COUNT 159

DELETE 196

DELETED() 200

EDIT 228

EOF() 238

FLUSH 268

GO 294

INSERT 309

INSERT AUTOMEM 310

INSERT BLANK 309

LUPDATE() 350

PACK 389

RECALL 423

RECCOUNT() 424

RECNO() 425

RECSIZE() 426

RELEASE AUTOMEM 433

REPLACE 439

REPLACE AUTOMEM 441

REPLACE FROM

ARRAY 445

SET AUTOSAVE 481

SET CARRY 484

SET DELETED 501

SKIP 578

STORE AUTOMEM 595

ZAP 630

record counters 540

comparing 122

record indicator *See* record
pointers

record numbers 4, 423

browse objects 938

display, suppressing 206

returning 425

record pointers 309

moving 294, 471, 481, 578

events and 885

linked tables 560

position, returning 103, 238

work areas 476

- records 1006
 - accessing sequentially 274
 - adding 70, 72, 309, 310, 484, 766, 925, 983
 - arrays and 75
 - events and 857
 - restricting 763
 - blank 71, 309
 - return values 93
 - browsing 230
 - changing 833
 - browse objects 814, 845
 - specific 440
 - combining 321
 - comparing multi-table 322
 - compressing 229
 - copying 73, 154, 445
 - automatically 143
 - counting 111, 204, 210, 424
 - deleting 196, 200, 389, 630, 757, 981
 - confirming 558
 - controlling 501
 - marking prevented 108, 231, 789, 936
 - displaying 206
 - editing 105, 107, 121
 - filling with blanks 102
 - fixed-length 143
 - locking 344, 461
 - information, getting 141, 338
 - retry messages 557
 - manipulating 581
 - moving between 109
 - Table Editor 232
 - moving through 295
 - processing 501, 516, 527
 - random 288
 - size, returning 426
 - stepping through 470
 - unlocking 552, 618
 - unmarking 423
 - updating 442
- RECSIZE() 426
 - DISKSPACE() and 205
 - RECCOUNT() and 424
- rectangle objects 740
 - adding borders 772
- recursive calls
 - DO 217
 - ON ERROR and 371
 - ON NETERROR and 378
- REDEFINE 427
- redefining colors 194
- referencing array elements 87, 948
- referencing files 8
- referencing objects 19, 830
 - forms 811
- referencing procedures and functions 17
- REFRESH 428
- Refresh() property 919
- refreshing screens 552, 919
- REINDEX 429
 - CONVERT and 142
 - INSERT and 309
 - REPLACE and 440
 - SET UNIQUE and 566
- RELATION() 430
- relational operators 20
- relationships (tables) 560
 - defining 554
 - restoring 430
- RELEASE 431
 - CLEAR MEMORY vs. 132
 - RELEASE AUTOMEM and 433
- RELEASE AUTOMEM 433
 - CLEAR MEMORY vs. 132
- RELEASE DLL 434
- RELEASE MENUS 435
- RELEASE OBJECT 435
- RELEASE POPUPS 436
- RELEASE SCREENS 436
- RELEASE WINDOWS 437
- Release() property 921
 - Reconnect and 919, 951
- releasing memory variables *See* clearing memory variables
- remainders (division) 360
- RemoveAll() property 922
- RemoveKey() property 922
- RENAME 437
- RENAME TABLE 438
- renaming files 437
- renaming tables 438
- repeating character strings 450
- repeating program
 - execution 220
- REPLACE 439
 - BLANK vs. 103
 - REPLACE AUTOMEM vs. 442
 - SET KEY and 528
- REPLACE AUTOMEM 441
- REPLACE BINARY 443
- REPLACE FROM ARRAY 445
- REPLACE MEMO 447
 - STORE MEMO and 596
- REPLACE MEMO...FROM 448
- REPLACE OLE 449
- replacement character
 - strings 599
- REPLICATE() 450
 - SPACE() vs. 586
- Report Designer 167, 169
- report files 170
- REPORT FORM 451
- reports
 - creating 170
 - displaying 452
 - formatting data 612
 - printing 233, 452
 - SET DESIGN and 502
- Resize() property 923
 - Grow() vs. 824
- resizing objects 847
- RESOURCE() 452
- RESTORE 453
 - SAVE and 469
- RESTORE IMAGE 454
- RESTORE SCREEN 456
- RESTORE WINDOW 456
- restoring memory variables 453
- restoring table relationships 430
- restricting data entry 502, 763, 875
- RESUME 456
 - SUSPEND vs. 603
- resuming program
 - execution 456, 458, 604
- retrieving data 984
- RETRY 457
- RETURN 458
 - QUIT vs. 417
- return codes (DOS) 417
- return values 301
 - absolute 48
 - angles 50, 84, 157, 577
 - tangents 90, 91, 608
 - averages 92, 111
 - blank records 93
 - characters as dates 179
 - decimal places 464
 - DLLs 97, 98, 99, 100, 101
 - hexadecimals 320
 - infinity 608
 - integers 313, 576
 - expressing equality 117, 267
 - maximum 111
 - minimum 111
 - modulus 360
 - seed values and 418
 - square roots 590
 - standard deviation 111
 - TYPE() 616
 - UDFs 458

- variance 111
- version numbers 627
- Windows API 97, 98, 99, 100, 101
- Right property 924
 - Bottom and 772
- RIGHT() 460
- RLOCK() 461
 - FLOCK() vs. 265
 - LOCK() vs. 344
 - SET REPROCESS and 557
 - UNLOCK and 618
- _rmargin 661
 - _alignment and 635
 - _wrap and 663
- ROLLBACK() 463
- rolling back transactions 463
- ROUND() 464
 - compared 313
- rounding 464, 598
- routines
 - See also* procedures
 - calling 458, 866
 - forms 383
 - returning control 458
- row selector *See* record pointers
- ROW() 465
- rows *See* records
- RTOD() 465
 - ACOS() and 51
 - ASIN() and 84
 - ATAN() and 90
 - ATN2() and 91
- RTRIM() 466
 - LTRIM() vs. 349
- RUN 467
 - ! vs. 41
 - DOS vs. 224
- RUN() 467
 - DOS vs. 224
 - RUN vs. 467
- running dBASE commands
 - page formatting 379
 - shortcuts 372, 519, 526
- running out of memory 604
- running programs *See* program execution
- running Windows
 - applications 467
- run-time errors 371
 - IDAPI 182, 184
 - line numbers, returning 333
 - messages, customizing 371, 509
 - multiuser environments 378
 - server 587, 589

S

- sample data 288
- SAVE 469
 - RESTORE and 453
- SAVE SCREEN 470
 - CLEAR SCREENS and 134
 - RELEASE SCREENS and 436
 - RESTORE SCREEN and 456
- SAVE TO clause 984
- SAVE WINDOW 470
- SaveRecord() property 925
- saving
 - data 268
 - automatically 481
 - multiuser environments 138
 - files 255
 - output 480
- ScaleFontName property 925
- ScaleFontSize property 926
- SCAN 470
 - EOF() and 238
- Scan() property 927
- scientific notation 241, 345, 440, 613
- scope 11
 - memory variables 10, 341, 399, 412, 591, 593
 - resetting 459
 - system memory variables 5, 459
- scope resolution operator 25
- screens
 - See also* displays
 - displays, slowing 235
 - refreshing 552
- scroll bars 742, 929, 965
- ScrollBar property 929
- search operations 11
 - arrays and 82, 927
 - case-insensitive 348, 620
 - case-sensitive 82, 89, 420, 928
 - pattern matching 332
- combo boxes 691
- conditions 342
- continuing 140
- data types, finding 324
- dates 475
- exact matches 260, 473
 - failing 539
- expressions, finding 346
- files 501, 542
 - checking existence 259
- key values and 304, 475, 528, 566
- matches, finding 274
- optimizing 260, 347, 474
- phonetic matches 202, 584
- sequential 260, 343, 473
- substrings 89, 419

- search order
 - files 218
 - preprocessor 674
- search path 218, 501, 542
 - DLL files 243
 - preprocessor 674
 - procedures and 405
- search-and-replace
 - operations 599
 - preprocessor 669, 676
 - string comparisons 511
- searching and summarizing
 - AVERAGE 92
 - CALCULATE 111
 - CONTINUE 140
 - DESCENDING() 201
 - FIND 259
 - FOUND() 274
 - KEYMATCH() 324
 - LOCATE 342
 - LOOKUP() 346
 - SEEK 473
 - SEEK() 475
 - SET NEAR 539
 - SUM 602
 - TOTAL 611
- secant 157
 - inverse 51
- SECONDS() 472
- security
 - ACCESS() 49
 - LOGOUT 346
 - PROTECT 409
 - SET ENCRYPTION 508
 - USER() 625
- seed values 417
- SEEK 473
 - EOF() and 238
 - FIND vs. 260
 - FOUND() and 274
 - INDEX and 305
 - LOCATE vs. 343
 - SET EXACT and 512
 - SET NEAR and 539
 - SOUNDEX() and 584
- SEEK() 475
 - EOF() and 238
 - FOUND() and 274
 - INDEX and 305
 - SEEK vs. 474
 - SET NEAR and 539
- SELECT 476, 984
- SELECT clause 983

- SELECT() 477
- SelectAll property 930
- Selected() property 931
 - DataSource and 787
 - Multiple and 849
- selecting
 - See also* choosing
 - colors 288
 - files 292
 - fonts 294, 816, 925
 - objects 804, 968
 - See also* focus, moving
 - prompts 714, 897
 - combo boxes 691
 - sessions 172
 - work areas 476, 477
- semicolons (;)
 - command separator 519
 - comment symbol 42
 - continuation character 406
- SEPARATOR parameter 559
- Separator property 931
- separators
 - command execution 519
 - date 535, 580
 - changing 496
 - decimal digits 545
 - directory paths 542
 - menus 931
 - thousands 559
 - time 580
- sequential access (data) 274
- sequential searches 260, 343, 473
- server errors 587, 589
- Server property 932
- ServerName property 933
- servers
 - See also* DDE server
 - applications; OLE server applications
 - connecting to 385, 725, 796, 831, 919
 - disconnecting 951
 - sessions, memory variables and 593
- SET 478
- SET... commands
 - changing interactively 478
 - CREATE SESSION and 173
 - current setting 569
 - information, getting 212
- SET ALTERNATE 479
 - ? command and 45
 - CLOSE ALTERNATE and 135
 - SET TALK and 562
- SET AUTOSAVE 481
- SET BELL 482
 - CHR() and 125
- SET BLOCKSIZE 483
 - overriding 537
 - SET IBLOCK and 522
 - SET MBLOCK vs. 537
- SET BORDER 484
- SET CARRY 484
 - APPEND and 71
- SET CATALOG 485
 - CATALOG() and 114
 - CREATE CATALOG and 162
 - SET TITLE and 564
- SET CENTURY 487
 - LUPDATE() and 350
 - YEAR() and 630
- SET clause 985
- SET COLOR OF 488
- SET COLOR TO 488
 - DEFINE COLOR and 194
- SET CONFIRM 489
- SET CONSOLE 489
 - overriding 489
- SET COVERAGE 490
 - #pragma vs. 675
- SET CUAENTER 492
- SET CURRENCY 493
- SET CURSOR 494
- SET DATABASE 495
 - BEGINTRANS() and 94
 - COMMIT() and 138
 - DATABASE() and 180
 - DIR and 204
 - DISPLAY FILES and 210
 - ISTABLE() and 319
 - ROLLBACK() and 463
- SET DATE 181, 496
 - LUPDATE() and 350
 - SET MARK and 535
- SET DATE TO 497
- SET DBTYPE 499, 979
 - CREATE and 160, 161
 - DIR and 204
 - DISPLAY FILES and 210
 - ISTABLE() and 319
- SET DECIMALS 500
 - ACOS() and 51
 - ASIN() and 84
 - ATAN() and 90
 - ATN2() and 91
 - AVERAGE and 93
 - COS() and 157
 - DTOR() and 226
 - EXP() and 241
 - FLOOR() and 267
 - FV() and 284
- LOG() and 345
- LOG10() and 345
- PAYMENT() and 392
- PI() and 395
- PV() and 416
- RANDOM() and 418
- ROUND() and 464
- RTOD() and 466
- SET PRECISION vs. 546
- SIGN() and 576
- SIN() and 577
- SQRT() and 590
- TAN() and 608
- SET DEFAULT 501
 - SET DIRECTORY and 505
- SET DELETED 501
 - CONVERT and 142
 - GO and 295
 - KEYMATCH() and 325
 - PACK vs. 389
 - RECALL and 423
 - RECNO() and 425
 - ShowDeleted and 937
- SET DELIMITERS 502
- SET DESIGN 502
- SET DEVELOPMENT 503
- SET DEVICE 504
 - CLOSE ALL and 135
 - PCOL() and 393
 - PROW() and 411
 - SET ALTERNATE vs. 479
 - SET TALK and 562
- SET DIRECTORY 504
 - CD vs. 115
- SET DISPLAY 506
- SET ECHO 506
- SET EDITOR 506
- SET ENCRYPTION 508
- SET ERROR 509
- SET ESCAPE 510
 - ON ESCAPE and 373
 - WAIT and 628
- SET EXACT 21, 511
 - ASCAN() and 82
 - FIND and 260
 - LIKE() and 332
 - LOCATE and 343
 - Scan() and 928
 - SEEK and 474
 - SEEK() and 475
 - SET KEY and 528
- SET EXCLUSIVE 513
 - FLOCK() vs. 265
- SET FIELDS 514
 - CLEAR FIELDS and 131
 - COPY and 144
 - COPY STRUCTURE and 150

- FLDLIST() and 263
- JOIN and 322
- SET CARRY vs. 485
- SET FILTER 516
 - GO and 295
 - KEYMATCH() and 325
 - RECNO() and 425
 - SET KEY and 528
- SET FORMAT 518
 - APPEND and 71
 - CLEAR PROGRAM and 133
 - COMPILE vs. 140
- SET FULLPATH 518
 - _dbwinhome and 640
 - DBF() and 183
 - HOME() and 297
 - MDX() and 353
 - NDX() and 368
- SET FUNCTION 519
 - FKLABEL() and 261
- SET HEADINGS 520
 - DISPLAY and 206
- SET HELP 521
- SET IBLOCK 522
 - SET BLOCKSIZE vs. 483
- SET INDEX 524
 - FLUSH and 268
 - REINDEX and 429
 - REPLACE and 440
 - SET EXCLUSIVE and 513
 - SET ORDER and 541
 - TAG() and 605
- SET INTENSITY 526
- SET KEY 526
 - FKLABEL() and 261
 - KEYMATCH() and 325
 - ON KEY and 375
- SET KEY TO 527
 - SET KEY and 526
- SET LDCHECK 529
- SET LDCONVERT 530
- SET LIBRARY 531
 - CLEAR PROGRAM and 133
- SET LOCK 532
- SET MARGIN 534
 - _ploffset and 656
- SET MARK 535
 - LUPDATE() and 350
 - SET DATE and 496
- SET MBLOCK 536
 - SET BLOCKSIZE vs. 483
- SET MEMOWIDTH 538
 - MLINE() and 360
- SET MESSAGE 539
- SET MOUSE 539
- SET NEAR 539
 - FOUND() 275
- SEEK and 474
- SET RELATION and 555
- SET ODOMETER 540
- SET ORDER 541
 - ORDER() and 387
- SET PATH 542
 - CD vs. 115
 - ERASE and 240
 - FSIZE() and 280
 - FTIME() and 281
 - ISTABLE() and 319
 - SET DIRECTORY vs. 505
- SET PCOL 543
- SET POINT 545
- SET PRECISION 546
- SET PRINTER 547
 - ? command and 45
 - CHOOSEPRINTER() and 124
 - CLOSE ALL and 135
 - CLOSE PRINTER and 135
 - PCOL() and 393
 - _pcolno and 647
 - PRINTJOB and 397
 - PROW() and 411
- SET PROCEDURE 549
 - CLEAR PROGRAM and 133
 - COMPILE vs. 140
 - SET LIBRARY vs. 531
- SET PROW 551
- SET REFRESH 552
- SET RELATION 553
 - CALCULATE and 112
 - COPY STRUCTURE and 151
 - FLOCK() and 266
 - FOUND() and 275
 - JOIN vs. 322
 - LOOKUP() and 347
 - RELATION() and 430
 - RLOCK() and 462
 - SET DELETED and 501
 - SET SKIP and 560
 - TARGET() and 609
 - UNLOCK and 618
- SET REPROCESS 557
 - FLOCK() and 266
 - RLOCK() and 462
- SET SAFETY 558
 - COPY BINARY and 146
 - COPY FILE and 147
 - COPY MEMO and 149
 - COPY STRUCTURE and 150
 - FCREATE() and 249
 - RENAME and 437
 - SAVE and 469
 - SET ALTERNATE and 480
 - STORE and 594
- TYPE and 615
- ZAP and 630
- SET SEPARATOR 559
 - DIR/DIRECTORY and 559
- SET SKIP 560
 - SET RELATION and 556
- SET SPACE 561
 - ? command and 45
- SET STEP 562
- SET TALK 562
 - AVERAGE and 93
 - CALCULATE and 112
 - CONTINUE and 141
 - LOCATE and 343
 - SET SAFETY and 558
- SET TIME 563
- SET TITLE 564
 - SET CATALOG and 486
- SET TOPIC 565
- SET TYPEAHEAD 566
- SET UNIQUE 566
 - INDEX and 305
 - REINDEX and 429
 - UNIQUE() and 617
- SET VIEW 567
 - CREATE VIEW...FROM ENVIRONMENT and 175
- SET WINDOW OF MEMO 568
 - overriding 568
- SET() 569
 - SETTO() vs. 569
- SET...TO commands, current setting 571
- SetFocus() property 934
- setting DDE links 831, 919
- SETTO() 571
- shape objects 745, 935
 - borders 909, 910
- ShapeStyle property 745, 935
- shared data commands
 - BEGINTRANS() 94
 - CHANGE() 122
 - COMMIT() 138
 - CONVERT 141
 - FLOCK() 265
 - ID() 298
 - LKSYS() 338
 - LOCK() 344
 - NETWORK() 369
 - ON NETERROR 378
 - RLOCK() 461
 - ROLLBACK() 463
 - SET EXCLUSIVE 513
 - SET LOCK 532
 - SET REFRESH 552
 - SET REPROCESS 557
 - UNLOCK 618

- shared mode 513
- SHELL() 573
- shift bits operators 98, 99
- Shift-key combinations
 - See also* keyboard; keystrokes
 - command execution 375, 519
- ShortCut property 936
- SHOW MENU 574
- SHOW OBJECT 575
- SHOW POPUP 576
- ShowDeleted property 936
- ShowHeading property 937
- ShowRecNo property 938
- ShowSpeedTip property 939
- SIGN() 576
- similar spellings, finding 584
- SIN() 577
 - ASIN() and 84
 - DTOR() and 226
 - PI() and 395
- sine 577
 - inverse 84
 - reciprocal 578
- single-line comments 42, 43
- Size property 940
- Sizeable property 940
 - OnSize and 900
- SKIP 578
 - EOF() and 238
 - FIND vs. 260
 - SCAN and 471
 - SEEK and 473
 - SEEK() and 475
 - SET KEY and 528
- SLEEP 580
 - interrupting 580, 581
- SORT 581
 - ASORT() vs. 85
 - INDEX vs. 305
- sort order 85
 - default 582
 - indexes 304
- Sort() property 941
- Sorted property 942
- sorting array elements 84, 941
- sorting data 85, 529, 581
 - See also* indexing and sorting
 - combo boxes 942
 - indexing vs. 583
 - list boxes 942
 - maximums and limits 1003
 - multiple fields 582
- sorting dates 227
- sound applications 796
- sound effects 145
 - playing 396
- SOUNDEX() 584
 - DIFFERENCE() and 203
- source code *See* code
- space characters 586
- SPACE() 586
 - REPLICATE() vs. 450
- spaces
 - leading 331
 - deleting 349
 - trailing, deleting 466, 613
- speakers 145, 482
- specifying Help topics 565
- specifying text in forms 944
- SpeedBar buttons 943
- SpeedBar property 943
- SpeedTip property 944
- spin boxes 746
 - formatting text 911
 - ranges, setting 915, 916, 917
 - text, changing 945
 - values, changing 930, 946
- spinners *See* spin boxes
- SpinOnly property 945
- spreadsheets 302
- SQL data types 980
- SQL databases 15
 - BEGINTRANS() and 94
 - indexing 199, 303, 528, 541
 - key expressions, finding 324
 - master index, specifying 622
 - linking tables 554
 - moving to specific records 104
 - statements, executing 588
 - structures, changing 362
- SQL, local commands 977–986
- SQLERROR() 587
- SQLEXP() 588
- SQLMESSAGE() 589
- SQRT() 590
- square root, returning 590
- standalone applications 573, 707
- standard classes 6
- standard deviation, returning 111
- statements 3, 6
 - See also* code
 - grouping 17
 - literal 17
- STATIC 591
- static variables 591
 - See also* memory variables
- scope, resetting 459
- statistical operations 111
- status bars
 - displaying messages 539, 945, 962
 - messages 562
 - record counters 540
- StatusMessage property 945
- Step property 946
- stepping through programs 186
- stepping through records 470
- stopping program
 - execution 113, 417, 458, 603
- STORE 593
 - implicit 4
- STORE AUTOMEM 595
 - CLEAR AUTOMEM and 129
 - RELEASE AUTOMEM and 433
 - STORE MEMO vs. 596
- STORE MEMO 596
 - REPLACE MEMO and 447
 - STORE vs. 594
- storing graphics
 - binary fields 146
 - memo fields 149
- storing text 145
- STR() 598
 - YEAR() and 630
- streaming output
 - See also* I/O
 - writing to files 547
- string comparisons 21, 350, 358
 - case sensitivity 351, 358
 - expressing equality 511
 - pattern matching 331
 - phonetic matching 202, 584
- string conversions
 - characters to dates 179, 363
 - characters to numbers 625
 - dates to characters 216, 225, 227, 353, 363
 - lowercase to uppercase 304, 620
 - first letter 408
 - numbers to strings 598
 - OEM characters to ANSI 69
 - uppercase to lowercase 348
- string data commands
 - ANSI() 69
 - AT() 89
 - CENTER() 118
 - DIFFERENCE() 202
 - ISALPHA() 314
 - ISLOWER() 317
 - ISUPPER() 320
 - LEFT() 328
 - LEN() 330
 - LIKE() 331

- LOWER() 348
- LTRIM() 349
- PROPER() 408
- RAT() 419
- REPLICATE() 450
- RIGHT() 460
- RTRIM() 466
- SET EXACT 511
- SOUNDEX() 584
- SPACE() 586
- STUFF() 599
- SUBSTR() 601
- TRANSFORM() 612
- TRIM() 613
- UPPER() 620
- string operators 22
 - substring comparison 21
- strings
 - assigning to keystrokes 519
 - duplicating 450
 - empty 225, 330, 511
 - expressions and 82, 928
 - leading spaces, deleting 349
 - literal 13
 - nesting 13
 - padding 598
 - processing 274
 - replacing specific
 - characters 599
 - returning 450, 599, 615
 - case 314, 317, 320
 - centered 118
 - dates 116, 137, 610
 - current 181
 - DLLs 452
 - formatted 612
 - number of characters 328, 330, 460
 - spaces 586
 - substrings 89, 419, 601
 - subscripts 681
 - trailing spaces, deleting 466, 613
 - writing to files 287
- structure-extended tables 156
 - creating 178
- structures *See* control structures; table structures
- STUFF() 599
- Style property 947
- subclasses 6
- subroutines *See* functions; procedures
- Subscript() property 948
 - Element() vs. 803
 - Scan() and 928
- subscripts 44, 681, 812
 - array *See* array elements
 - substituting memory
 - variables 977
 - SUBSTR() 601
 - substring operator 21
 - substrings
 - finding 89, 419
 - replacing characters 599
 - returning 601
 - subtracting dates 20
 - subtraction operator 20
 - SUM 602
 - SET HEADINGS and 520
 - TOTAL vs. 612
 - superscripts 44
 - SUSPEND 603
 - PROGRAM() and 407
 - suspending program
 - execution 113, 603
 - for specified duration 580
 - syntax 7, 27
 - codeblocks 18
 - errors 140
 - NEW operator 22
 - SysMenu property 949
 - system
 - bell, setting 482
 - clock 181, 610
 - setting 497, 563
 - time elapsed 235, 472
 - date
 - changing 496
 - returning 181
 - system memory variables 4, 397, 459
 - _alignment 635
 - _app 636
 - _box 638
 - _curobj 639
 - _dbwinhome 640
 - _indent 640
 - _lmargin 642
 - _padvance 643
 - _pageno 644
 - _pbpage 645
 - _pcolno 646
 - _pcopies 647
 - _pdriver 648
 - _pject 649
 - _pepage 650
 - _pform 651
 - _plength 653
 - _plineno 654
 - _ploffset 656
 - _porientation 656
 - _ppitch 657
 - _pquality 658
 - _pspacing 659
 - _rmargin 661
 - _tabs 662
 - _wrap 663
- system utilities and information
 - CD 115
 - _dbwinhome 640
 - DIR/DIRECTORY 203
 - DISKSPACE() 205
 - DOS 223
 - GETENV() 290
 - HOME() 297
 - MD 352
 - MKDIR 359
 - OS() 388
 - RUN 467
 - RUN() 467
 - SET DEFAULT 501
 - SET DIRECTORY 504
 - SET PATH 542
 - VALIDDRIVE() 626

T

- tab boxes
 - prompts
 - current 784
- Tab key 492, 950
- tabbing order 639, 765, 852
 - See also* focus
 - SpeedBar buttons 943
- table basics commands
 - ALIAS() 68
 - APPEND FROM 73
 - CATALOG() 114
 - CLOSE ALL 135
 - CLOSE DATABASES 135
 - CLOSE TABLES 135
 - COPY 143
 - COPY STRUCTURE 150
 - COPY TABLE 151
 - COPY TO...STRUCTURE
 - EXTENDED 155
 - CREATE 160
 - CREATE CATALOG 162
 - CREATE...FROM 175
 - CREATE...STRUCTURE
 - EXTENDED 177
 - DATABASE() 180
 - DBF() 183
 - DELETE TABLE 198
 - DISPLAY STRUCTURE 214
 - IMPORT 302
 - ISTABLE() 319
 - LIST STRUCTURE 334
 - MODIFY STRUCTURE 362
 - OPEN DATABASE 385
 - REFRESH 428

- RENAME TABLE 438
- SELECT 476
- SELECT() 477
- SET CATALOG 485
- SET DATABASE 495
- SET DBTYPE 499
- SET TITLE 564
- SQLEXEC() 588
- USE 621
- WORKAREA() 629
- Table Designer 161, 362
- Table Editor
 - activating 109, 232
 - colors, setting 106, 229
 - display, compressing 229
- Table Expert 161
- table names
 - aliases 9
 - changing 438
 - default 144
 - returning 183, 609
- table organization commands
 - filters *See* filters, queries, and views
 - indexing *See* indexing and sorting
 - linking *See* linking and relating
 - queries *See* filters, queries, and views
 - relations *See* linking and relating
 - searching *See* searching and summarizing
 - sorting *See* indexing and sorting
 - summarizing *See* searching and summarizing
 - views *See* filters, queries, and views
- table structures 1005
 - catalogs 163
 - changing 161, 362
 - copying 150, 155, 176
 - designing 177
 - displaying 214
 - modifying 978
 - predefined 162
 - storing 59, 808
- tables
 - accessing, browse objects 760
 - adding fields 363, 978
 - adding records 71, 72, 309, 310, 484, 983
 - events and 857
 - restricting 763
 - aliases 760
 - changing 160
 - closing 135, 136, 346
 - work areas 135
 - controlling access 409
 - copying 143, 151
 - creating 143, 150, 155, 175, 321, 582, 979
 - related 151
 - temporary 611
 - default type 499
 - deleting 198, 982
 - deleting fields 978
 - deleting indexes 982
 - deleting records 981
 - directory listings 203
 - encrypting 508
 - existing 319
 - exporting with COPY 144
 - importing 302
 - indexing *See* indexes
 - information, getting 59, 212, 808
 - linking 553
 - See also* links
 - locking 265, 532
 - information, getting 141, 338
 - retry messages 557
 - maximums and limits 1001
 - moving through 294
 - linked 560
 - opening 9, 478, 621
 - file-sharing modes 513
 - Paradox *See* Paradox tables
 - relations 560
 - defining 554
 - restoring 430
 - retrieving data 984
 - SET DESIGN and 502
 - size, returning 424
 - structure-extended 156, 178
 - temporary 611
 - unlocking 552, 618
 - updating data 985
- _tabs 662
- tabs 751, 763
- TabStop property 950
- TAG() 605
 - SET INDEX and 525
- TAGCOUNT() 606
- TAGNO() 607
 - FOR() AND 271
- TAN() 608
 - DTOR() and 226
 - PI() and 395
- tangent 608
 - inverse 90, 91
 - reciprocal 608
- TARGET() 609
- TEDIT setting 164
- template characters 559, 612, 820, 911
- temporary files 283, 623
 - SORT and 582
- temporary tables 611
- Terminate() property 951
 - Initiate() and 831
 - Reconnect and 919
- testing case 314, 317, 320
- testing conditions 272, 300
- testing memory variables 272
- testing programs 120, 288, 490
- TEXT 610
- text
 - See also* memo fields
 - centering 118
 - copying 779, 800
 - deleting 785, 801
 - fonts 815, 818, 819
 - formatting 44, 820
 - size 817, 926
 - with spaces 586
 - losing 597
 - offset margins 534
 - overstriking 44, 45, 818
 - paging through 929
 - pasting 801, 906
 - SET DESIGN and 502
 - specifying 944
 - spin boxes 945
 - storing 145
 - underlining 44, 819
 - wordwrapping 972
- Text Editor
 - activating 164
 - memo fields 538, 697
- text editors, alternate 164
 - specifying 507, 697
- text files
 - closing 135
 - copying to memo fields 77, 448
 - creating 506
 - displaying 614
 - editing 165, 697
 - printing 614
 - writing to 206, 614
 - alternate 479
 - environment
 - information 212, 562
 - files list 209
 - from memo fields 149
 - streaming output 547
- text objects 753
 - aligning graphics 761

- backgrounds 906
- colors, setting 778
- formatting text 911
- Text property 952
 - DisabledBitmap and 795
 - DownBitmap and 798
 - FocusBitmap and 813
 - UpBitmap and 960
- text strings *See* strings
- This variable 126
- thousands separator 559
- time 497, 610
 - See also* clock
 - default settings 496, 535
 - elapsed 235, 472
 - resetting 563
 - separators 580
- time commands *See* date and time commands
- time formats 563, 610
 - SLEEP 580
 - specifying 496
- time stamps 497, 563
 - returning 281
- TIME() 610
 - ELAPSED() and 235
 - SECONDS() vs. 472
 - SET TIME and 563
- Timeout property 953
- Toggle property 953
- Top property 954
 - Bottom and 772
 - Left and 836
 - Right and 924
- Topic property 955
- TopMost property 956
- TOTAL 611
 - SUM vs. 603
- TrackRight property 958
- trailing spaces, deleting 466, 613
- trailing zeros, deleting 464
- transactions 94
 - committing 138
 - DDE applications 953
 - rolling back 463
- TRANSFORM() 612
 - SET CURRENCY and 493
- trigonometric functions
 - ACOS() 50
 - ASIN() 84
 - ATAN() 90
 - ATN2() 91
 - COS() 157
 - DTOR() 226
 - RTOD() 465
 - SIN() 577
 - TAN() 608

- TRIM() 613
 - LTRIM() vs. 349
 - RTRIM() vs. 466
- truncating numeric data 313
- two-dimensional arrays *See* arrays
- TYPE 614
- type conversions
 - ASC() 81
 - CHR() 124
 - CTOD() 179
 - DTOD() 225
 - DTOS() 227
 - EMPTY() 237
 - HTOI() 297
 - ITOH() 320
 - STR() 598
 - VAL() 625
- TYPE() 615
- typeahead buffer 324
 - clearing 134
 - information, getting 306, 369
 - size, setting 566
- types
 - See also* data types
 - field, returning 615
- typographical conventions 2, 7

U

- UDFs 4
 - accessing 531
 - coverage analysis 208, 490
 - debugging 185, 186, 406
 - declaring 282
 - executing 519
 - execution, stopping 458
 - return values 458
 - running 549
- Unadvise() property 958
- unallocated memory,
 - clearing 268
- unary operators 20
- #undef 676
- undefined identifiers 669, 676
- underlining text 44, 819
- Undo() property 959
- undoing changes, multiuser
 - environments 463
- unique indexes 566, 617
- UNIQUE() 617
- UNLOCK 618
- unlocking files 552, 618
- unmarking records 423
- unrelated tasks 386
- unsupported language
 - elements 998
- UpBitmap property 960
 - DownBitmap and 798
- UPDATE 619, 985
- UPDATED() 620
- updating arrays 79
- updating catalogs 486
- updating data 350, 619
 - multiuser environments 138, 265, 461
- updating data buffers 428
- updating indexes 363, 429, 566
 - APPEND and 71
 - automatically 305
 - EDIT and 231
 - INSERT and 309
- UPPER() 620
 - ASCAN() and 82
 - AT() and 89
 - INDEX and 304
 - LIKE() and 332
 - RAT() and 420
 - Scan() and 928
- uppercase letters 304, 408, 620
 - converting to lowercase 348
 - sorting data 582
 - testing 320
- USA date format 496
- USE 9, 621
 - ALIAS() and 69
 - FLUSH and 268
 - REINDEX and 429
 - SET INDEX and 524
 - TAG() and 605
- USE...AUTOMEM
 - APPEND AUTOMEM and 72
 - RELEASE AUTOMEM and 433
 - STORE AUTOMEM vs. 595
- USE...EXCLUSIVE
 - CONVERT and 142
 - FLOCK() vs. 265
 - NETWORK() and 369
 - SET EXCLUSIVE and 513
- user choices, evaluating 336, 782
- user names, getting 298
- USER() 625
- user-defined functions *See* UDFs
- user-defined memory
 - variables 131
- user-defined types, binary 145
- user-defined windows 6

V

- VAL() 625
- Valid property 961

- OnLostFocus vs. 875
- ValidErrMsg and 962
- ValidRequired and 963
- validating data 963
- VALIDDRIVE() 626
 - CD and 115
 - SET DIRECTORY and 505
- ValidErrMsg property 962
- ValidRequired property 963
- Value property 964
 - SelectAll and 930
- values 10, 12
 - absolute 48
 - arrays 93, 112
 - assigning to arrays 60, 64, 188, 810
 - averaging 92, 111
 - blank 102
 - calculations 112
 - comparing
 - logical 350, 358
 - multi-table 322
 - decimal 297
 - converting to
 - hexadecimal 320
 - keystrokes, returning 306, 369
 - duplicate
 - checking 325
 - keys 560
 - entry fields, changing 930
 - finding 347, 475
 - float *See* float values
 - literal 9
 - dates 14
 - memory variables 92, 112
 - finding 474
 - monetary 493
 - net present 111
 - retaining in memory 591
 - return *See* return values
 - returning from UDFs 458
 - seed, defined 417
 - specifying specific 301, 668
 - spin boxes 917, 930, 946
 - setting 915, 916
 - zero *See* zero values
- VALUES clause 983
- variable-length fields 15
- variables *See* memory variables; system variables
- variance 111
- VARREAD() 627
- version control, programs 670, 674
- version numbers, returning 388, 627

- VERSION() 627
- Vertical property 965
- vertical scroll bars 965
- view files 175
- View property 965
- viewing
 - See also* displaying
 - field names in browse objects 937
 - objects 967
 - programs 45
 - executing 186
 - property settings 312
- views 791
- Visible property 967
- volume, measuring 395

W

- WAIT 628
 - SLEEP vs. 580
- warning beeps 125, 482
- warnings 962
 - See also* messages
- When property 968
 - OnClick and 862
 - OnGotFocus vs. 865
 - OnSize and 900
- WHERE clause 981, 984, 985
- whole numbers 313, 464
 - See also* integers
 - thousands separator 559
- Width property 969
 - Alignment vs. 762
 - Height and 826
- widths *See* display widths
- wildcard characters
 - directory listings 9, 204, 210
 - fields list 516
 - pattern matching 332
 - temporary files 283
- Window menu, adding to
 - forms 970
- WINDOW() 629
- WindowMenu property 970
- windows 704
 - BROWSE *See* Table Editor
 - clearing 128
 - Command *See* Command window
 - displaying 580
 - EDIT *See* Table Editor
 - modeless *See* non-modal forms
 - setting 568
 - user-defined 6
- Windows 95 247, 250, 269, 279, 991
 - file attributes 794
- Windows 95 commands
 - FACCESSDATE() 247
 - FCREATEDATE() 250
 - FCREATETIME() 250
 - FNAMEMAX() 269
 - FSHORTNAME() 279
- Windows API 298, 321
 - return values 97, 98, 99, 100, 101
- Windows applications, running 467
 - See also* applications
- Windows Multiple Document Interface 841
- Windows programming commands
 - BITAND() 97
 - BITLSHIFT() 98
 - BITOR() 99
 - BITRSHIFT() 99
 - BITSET() 100
 - BITXOR() 101
 - EXTERN 242
 - HELP 296
 - LOAD DLL 340
 - RELEASE DLL 434
 - RESOURCE() 452
 - SET HELP TO 521
 - SET TOPIC 565
 - SET TOPIC TO 565
- Windows Sound Recorder 396
- WindowState property 971
- wordwrap editor 1003
- wordwrapping text 972
- work areas
 - active indexes 606
 - aliases 9, 623
 - returning 68, 477
 - catalog files 162, 163, 486
 - closing files 135
 - closing tables 135
 - current, returning 629
 - data buffers 428
 - maximums and limits 1003
 - naming 69
 - opening files 524
 - opening tables 9, 623
 - record pointers 476
 - returning tables 183
 - search operations 274
 - selecting 476, 477
- WORKAREA() 629
- working directory, current *See* directories

working drive, current *See* disk drives
working environment 175
 See also views; work areas
_wrap 663
 _alignment and 635
 _indent and 640
 _tabs and 662
Wrap property 972

X

XOR bitwise operator 101

Y

YEAR() 630

Z

ZAP 630
 DELETE vs. 197
 PACK vs. 389
 RECALL and 423
 SET SAFETY and 558
zero values
 blank vs. 103
 equality comparisons 267
 finding 576
 random numbers 418
 trailing, deleting 464