Quattro® Pro for Windows

Building Spreadsheet Applications

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

Copyright © 1987, 1992 by Borland International. All rights reserved. Quattro is a registered trademark of Borland International, Inc. Other brand and product names are trademarks or registered trademarks of their respective holders.

Fontware is a trademark of Bitstream, Inc., except that in the United Kingdom, France, and West Germany, the mark Fontware is licensed to Bitstream by Electronic Printing Systems, Ltd.

R2 W H

10 9 8 7 6 5 4 3 2 1

C O N T E N T S

Introduction 1

Chapter 1 Using @functions 3

- Entering @functions 4
  - @Function features 4
  - Syntax rules 5
  - Nesting @functions 5
  - Comments and @functions 6
  - Argument types 6
  - Operators 7
    - Operator precedence 7
    - Operator use 8
  - ERR and NA 9
- Functions by type 9
  - Mathematical @functions 10
  - Statistical @functions 10
  - Database @functions 12
  - Logical @functions 13
  - Financial @functions 13
    - Annuity @functions 14
    - Cash flow @functions 16
    - Depreciation @functions 16
  - Date and time @functions 17
  - String @functions 18
  - Miscellaneous @functions 19
- Add-in @functions 20

Chapter 2 @Functions reference 21

- @@ 21
- @ABS 22
- @ACOS 22
- @ASIN 23
- @ATAN 23
- @ATAN2 23
- @AVG 24
- @CELL 25
- @CELLINDEX 27
- @CELLPOINTER 27
- @CHAR 28
- @CHOOSE 29
- @CLEAN 30
- @CODE 30
- @COLS 30
- @COMMAND 30
- @COS 31
- @COUNT 32
- @CTERM 32
- @CURVALUE 34
- @DATE 34
- @DATEVALUE 35
- @DAVG 36
- @DAY 37
- @DCOUNT 38
- @DDB 39
- @DDELINK 40
- @DEGREES 41
- @DMAX 42
- @DMIN 43
- @DSTD 44
- @DSTDS 45
- @DSUM 46
- @DVAR 47
- @DVARS 48
- @ERR 49
- @EXACT 49
- @EXP 50
- @FALSE 50
- @FILEEXISTS 50
- @FIND 51
- @FV 52
- @FVAL 53
- @HEXTONUM 54
- @HLOOKUP 54

@HOUR ... 56
@IF ... 57
@INDEX ... 58
@INT ... 59
@IPAYMT ... 60
@IRATE ... 61
@IRR ... 62
Simple transactions ... 62
Multiple rates of return ... 63
@ISERR ... 65
@ISNA ... 66
@ISNUMBER ... 66
@ISSTRING ... 67
@LEFT ... 67
@LENGTH ... 68
@LN ... 68
@LOG ... 69
@LOWER ... 69
@MAX ... 69
@MEMAVAIL ... 70
@MEMEMSAVAIL ... 70
@MID ... 70
@MIN ... 71
@MINUTE ... 71
@MOD ... 72
@MONTH ... 72
@N ... 73
@NA ... 74
@NOW ... 75
@NPER ... 75
@NPV ... 76
@NUMTOHEX ... 78
@PAYMT ... 78
@PI ... 79
@PMT ... 79
@PPAYMT ... 81
@PROPER ... 82
@PROPERTY ... 82
@PV ... 83
@PVAL ... 84
@RADIANS ... 85
@RAND ... 85
@RATE ... 86
@REPEAT ... 87
@REPLACE ... 87
@RIGHT ... 88
@ROUND ... 89
@ROWS ... 89
@S ... 90
@SECOND ... 90
@SHEETS ... 91
@SIN ... 91
@SLN ... 92
@SQRT ... 92
@STD ... 93
@STDS ... 94
@STRING ... 94
@SUM ... 95
@SUMPRODUCT ... 96
@SYD ... 96
@TAN ... 97
@TERM ... 98
@TIME ... 99
@TIMEVALUE ... 99
@TODAY ... 100
@TRIM ... 100
@TRUE ... 101
@UPPER ... 101
@VALUE ... 101
@VAR ... 102
@VARS ... 103
@VERSION ... 103
@VLOOKUP ... 103
@YEAR ... 105

Chapter 3 Using macros 107
What is a macro? ... 107
Recording macros ... 108
Basic macro recording ... 108
Recording modes ... 109
Standard addresses vs. relative references ... 110
Recording tips ... 110
Macro libraries ... 110
Running macros ... 111
Suppressing screen redraw ... 112
Running macros from Windows ... 112
Attaching a macro to a key ... 113

Attaching macros to notebook buttons . . . 113
Attaching macros to graph buttons . . . 115
Assigning macros to dialog box objects and menu commands . . . 115
Autoload macros . . . 115
Quattro Pro for DOS macros . . . 116
1-2-3 macros . . . 116
Typing macros . . . 117
The basic procedure . . . 117
Block names and macros . . . 117
Macro tips . . . 118
Flow charts . . . 118
Typing tips . . . 118
Macro commands . . . 119
Macro syntax . . . 120
Arguments in macro commands . . . 121
Entering macro commands . . . 122
Keyboard . . . 123
Screen . . . 124
Interactive . . . 124
Program Flow . . . 125
Subroutines . . . 125
Arguments and subroutines . . . 126
Cell . . . 127
File . . . 128
Command Equivalents . . . 129
Dynamic Data Exchange (DDE) . . . 131
Quattro Pro as a DDE server . . . 132
UI Building . . . 133
Object . . . 134
Creating and positioning objects . 135
Selecting, positioning, and sizing objects . . . 135
Changing/reading property settings . . . 136
Accessing other notebooks . . . 136
Relative references . . . 137
Self-modifying macros . . . 138
Debugging macros . . . 139
The debug window . . . 139
The debugger menu . . . 140
Stepping through a macro . . . 140
Setting breakpoints . . . 140
Standard breakpoints . . . 141
Breakpoints in action . . . 141
Conditional breakpoints . . . 142
Monitoring cells . . . 143
Editing macro cells . . . 143
Clearing trace cells and breakpoints . . 144
Exiting Debug mode . . . 144

Chapter 4 Macro command reference 145
Macro commands by type . . . 145
The /x commands . . . 155
Macro command descriptions . . . 155
{ } . . . 156
{ ; } . . . 156
{ ? } . . . 157
{ABS} . . . 157
{ACTIVATE} . . . 158
{ADDMENU} . . . 159
{ADDMENUITEM} . . . 159
{Align.*Option*} . . . 161
{ALT} . . . 161
{Application.*Property*} . . . 162
{BACKSPACE} and {BS} . . . 165
{BACKTAB} . . . 166
{BEEP} . . . 166
{BIGLEFT} . . . 166
{BIGRIGHT} . . . 167
{BLANK} . . . 167
{BlockCopy} . . . 167
{BlockDelete.*Option*} . . . 168
{BlockFill.*Option*} . . . 168
{BlockInsert.*Option*} . . . 169
{BlockMove} . . . 169
{BlockMovePages} . . . 170
{BlockName.*Option*} . . . 170
{BlockReformat} . . . 171
{BlockTranspose} . . . 171
{BlockValues} . . . 171
{BRANCH} . . . 171
{BREAK} . . . 172
{BREAKOFF} . . . 172
{BREAKON} . . . 173
{CALC} . . . 173

{CAPOFF} and {CAPON} . . . 173
{CHOOSE} . . . 173
{CLEAR} . . . 173
{ClearContents} . . . 174
{CLOSE} . . . 174
{COLUMNWIDTH} . . . 174
{CONTENTS} . . . 176
{Controls.*Option*} . . . 177
{CR} or ~ . . . 178
{CREATEOBJECT} . . . 178
{CTRL} . . . 179
{DATE} . . . 179
{DEFINE} . . . 180
{DEL} and {DELETE} . . . 181
{DELETEMENU} . . . 181
{DELETEMENUITEM} . . . 182
{DialogView} . . . 182
{DialogWindow.*Property*} . . . 182
{DISPATCH} . . . 185
{DODIALOG} . . . 186
{DOWN} and {D} . . . 187
{EDIT} . . . 187
{EditClear} . . . 187
{EditCopy} . . . 188
{EditCut} . . . 188
{EditGoto} . . . 188
{EditPaste} . . . 188
{END} . . . 188
{ESC} and {ESCAPE} . . . 188
{EXEC} . . . 189
{EXECUTE} . . . 189
{ExportGraphic} . . . 190
{FileClose...} . . . 191
{FileCombine} . . . 191
{FileExit} . . . 191
{FileExtract} . . . 192
{FileImport} . . . 192
{FileNew} . . . 192
{FileOpen} . . . 192
{FileRetrieve} . . . 193
{FileSave...} . . . 193
{FILESIZE} . . . 193
{FLOATCREATE} . . . 194
{FLOATMOVE} . . . 195
{FloatOrder.*Option*} . . . 196
{FLOATSIZE} . . . 196
{FOR} . . . 197
{FORBREAK} . . . 198
{Frequency.*Option*} . . . 198
{FUNCTIONS} . . . 199
{GET} . . . 199
{GETDIRECTORYCONTENTS} . . . 200
{GETLABEL} . . . 200
{GETNUMBER} . . . 201
{GETOBJECTPROPERTY} . . . 202
{GETPOS} . . . 202
{GETPROPERTY} . . . 203
{GETWINDOWLIST} . . . 204
{GOTO} . . . 204
{GRAPH} . . . 204
{GRAPHCHAR} . . . 204
{GraphCopy} . . . 205
{GraphDelete} . . . 206
{GraphEdit} . . . 206
{GraphNew} . . . 206
{GRAPHPAGEGOTO} . . . 207
{GraphSettings.Titles} . . . 207
{GraphSettings.Type} . . . 207
{GraphView} . . . 208
{GraphWindow.*Property*} . . . 209
{Group.*Option*} . . . 210
{GroupObjects} . . . 210
{HELP} . . . 210
{HLINE} . . . 210
{HOME} . . . 211
{HPAGE} . . . 211
{IF} . . . 211
{IFKEY} . . . 212
{ImportGraphic} . . . 212
{INDICATE} . . . 212
{INITIATE} . . . 213
{INS}, {INSERT}, {INSOFF}, and {INSON} . . . 214
{InsertBreak} . . . 214
{InsertObject} . . . 214
{Invert.*Option*} . . . 215
{LEFT} and {L} . . . 215
{LET} . . . 216
{Links.*Option*} . . . 217

{LOOK} 217
{MACROS} 218
{MARK} 219
{MENUBRANCH} 219
{MENUCALL} 221
{MESSAGE} 222
{MOVETO} 223
{Multiply.*Option*} 223
{NAME} 224
{NamedStyle.*Option*} 224
{NEXTPANE} 225
{NEXTWIN} 225
{Notebook.*Property*} 225
{NUMOFF} and {NUMON} 227
{ONERROR} 227
{OLE.*Option*} 229
{OPEN} 229
{Optimizer.*Option*} 231
{Order.*Option*} 233
{Page.*Property*} 233
{PANELOFF} 236
{PANELON} 236
{Parse.*Option*} 237
{PasteFormat} 237
{PasteLink} 238
{PasteSpecial} 238
{PAUSEMACRO} 239
{PGDN} and {PGUP} 239
{POKE} 240
{Preview} 240
{Print.*Option*} 241
{PrinterSetup} 243
{PUT} 244
{PUTBLOCK} 245
{PUTCELL} 246
{QGOTO} 246
{QUERY} 246
{Query.*Option*} 247
{QUIT} 247
{READ} 248
{READLN} 249
{RECALC} 250
{RECALCCOL} 251
{Regression.*Option*} 252
{REQUEST} 253
{RESIZE} 254
{ResizeToSame} 255
{RESTART} 255
{RestrictInput.*Option*} 255
{RETURN} 256
{RIGHT} and {R} 256
{ROWCOLSHOW} 256
{ROWHEIGHT} 257
{SCROLLOFF} and {SCROLLON} 258
{Search.*Option*} 258
{SELECTBLOCK} 259
{SELECTFLOAT} 259
{SELECTOBJECT} 260
{Series.*Option*} 260
{SETGRAPHATTR} 262
{SETMENUBAR} 262
{SETOBJECTPROPERTY} 263
{SETPOS} 264
{SETPROPERTY} 264
{SHIFT} 265
{Slide.*Option*} 265
{SolveFor.*Option*} 266
{Sort.*Option*} 267
{SPEEDFILL} 267
{SPEEDFORMAT} 268
{SPEEDSUM} 268
{STEP} 268
{STEPOFF} 269
{STEPON} 269
{Subroutine} 269
{TAB} 270
{TABLE} 271
{TableQuery.*Option*} 271
{TableView} 271
{TERMINATE} 272
{UNDO} 272
{UngroupObjects} 272
{UP} and {U} 272
{VLINE} 273
{VPAGE} 273
{WAIT} 273
{WhatIf.*Option*} 274
{WINDOW} 275

{WindowArrIcon} 275
{WindowCascade} 275
{WindowClose} 275
{WindowHide} 276
{WindowMaximize} 276
{WindowMinimize} 276
{WindowMove} 276
{WindowNewView} 276
{WindowNext} 277
{WindowPanes} 277
{WindowRestore} 277
{WindowShow} 277
{WindowSize} 278
{WINDOWSOFF} 278
{WINDOWSON} 279
{WindowTile} 279
{WindowTitles} 279
{WorkSpace.*Option*} 280
{WRITE} 280
{WRITELN} 281
{ZOOM} 282

Chapter 5 Application basics 283
What's an application? 283
The developer vs. the user 284
Looking at a sample application 284
Using the Budgeteer 285
Creating a new budget file 286
Adding expenses to a database 286
Adding records 286
Viewing an expense database 287
Searching for specific records 287
The Budgeteer UI 288
The Budgeteer menus 288
The Budgeteer SpeedBar 288
Looking at how the application works 289
Understanding application components 291
Designing dialog boxes 291
Displaying dialog boxes 292
Using link commands 293
Designing SpeedBars 293
Displaying SpeedBars 294
Designing menus 294
Designing macros 296
Autoload macros 296
Changing properties with macro commands 296
Preparing to build an application 297
Planning an application 297
Design guidelines 297
Using the Developer mode 298
Where do I go from here? 299

Chapter 6 Dialog boxes and SpeedBars 301
An overview 301
Tools for building dialog boxes 302
The dialog window 303
The dialog window SpeedBar 304
General procedures 305
Procedures for creating a dialog box 305
Procedures for building a SpeedBar 305
Creating a sample dialog box 306
Opening a new dialog window 306
Adding controls to the dialog box 307
Labeling controls 311
Customizing controls 311
Using the {DODIALOG} command 312
Linking commands to controls 315
Testing the dialog box 317
Creating a sample SpeedBar 318
Opening a new SpeedBar 318
Adding controls to the SpeedBar 319
Linking commands to the controls 320
Displaying the SpeedBar 321
More about SpeedBars 322
Using Test mode 323
Working with dialog boxes 323
Working with controls 324
Providing hints 324
Control properties 324
Selecting several controls 324
Copying and moving controls 324
Copying parent controls 325
Moving and sizing controls 325
Using the Dimension options 325

Moving overlapping controls 326
Aligning controls 326
Aligning multiple controls 326
Repositioning controls 327
Setting a control's value 329
Setting a control's initial value 329
Changing the order in which you pass initial values 330
Editing the setting order 331
Changing a control's text 331
Using the Clipboard 331
Assigning a shortcut key to a control . 332
Setting tab order 333
Selection order within a group box . . . 333
Disabling and concealing controls 334
Grouping controls 335
Nesting grouped controls 336
Resizing grouped controls 336
Connecting controls to cells or other controls 339
Types of controls 340
Common button properties 340
Push buttons 340
Check boxes 341
Radio buttons 341
Bitmap buttons 341
Labels 342
Edit fields 344
Spin controls 345
Rectangles 346
Group boxes 346
List boxes 347
Combo boxes 347
Pick lists 348
File controls 349
Color controls 349
Scroll bars 350
Time controls 351
Working with Link commands 352
The Object Link dialog box 352
Link events 353
Assigning a link command to a control 355
Link command examples 355
Example 1: A link command that runs a macro 355
Example 2: A link command that changes a notebook's zoom factor . . 357
Example 3: A link command that resets the zoom factor to 100% 359
Example 4: A link command that reads the alignment of a specific cell 360
Example 5: A link command that displays another dialog box 361
Example 6: A link command that simulates events 363
A time control example 364
Dialog box examples 366
Example 1: Connecting a file control to an edit field 366
Example 2: Creating a list box 367
Example 3: Creating a history list (combo box) 368
Chapter 7 Menus 371
What's a menu? 371
Adding menus to the menu bar 373
Replacing the menu bar 374
Command definitions 374
Naming a menu command 375
Making menu commands act (Link commands) 375
EXECUTE 376
DOMACRO 376
RECEIVE 377
SEND 377
SET 377
Menu hints 377
Assigning a shortcut key 378
Dimming menu commands 378
Placing check marks by menu commands 379
Deleting menus 379
Adding and deleting menu commands . 379
Divider lines 380
Changing command properties 380
Designing a menu 382

Appendix A Command equivalents 383

Appendix B Property reference 399

Identifying objects 399
Object precedence 401
Objects and properties 402
Common properties 402
Color 402
Dimension 403
Font 403
Common graph object properties . . 404
Active objects and menu commands . 406
Dialog box objects 408
Graph objects (drawn) 417
Graph objects (fixed) 420
Notebook objects 426
Graphs page icons 428

Appendix C ANSI codes 431

Index 435

T A B L E S

1.1: @Function arguments 7
1.2: Quattro Pro formula operators 8
1.3: Arithmetic operators 8
1.4: Text operator 8
1.5: Logical operators 9
1.6: Mathematical @functions 10
1.7: Statistical @functions 11
1.8: Database @functions 12
1.9: Logical @functions 13
1.10: Financial @functions 14
1.11: Relationships among annuity @functions 15
1.12: Date and time @functions 17
1.13: String @functions 18
1.14: Miscellaneous @functions 19
3.1: Macro button properties 114
3.2: Items available from the topic System 133
4.1: Keyboard commands listed by category 146
4.2: Screen commands 147
4.3: Interactive commands 148
4.4: Program flow commands 148
4.5: Cell commands 149
4.6: UI Building commands 150
4.7: File commands 150
4.8: DDE commands 150
4.9: Object commands 151
4.10: Miscellaneous commands 151
4.11: Command equivalents 152
4.12: /x commands and the corresponding macro commands 155
4.13: Numeric format codes 177
4.14: Graph objects {CREATEOBJECT} can generate 178
4.15: OLE commands 229
4.16: *ReplaceOption* settings 244
4.17: Arguments for *DataToReceive* 254
5.1: Pages used in the Budgeteer 290
5.2: Budgeteer dialog boxes 290
5.3: Macro commands for defining a custom menu 295
5.4: Accessing properties in Developer mode 299
5.5: Special graph object properties 299
5.6: Pressing *Ctrl+Shift+N*, the Developer mode shortcut key 299
6.1: Editing a dialog box using its icon . . 323
6.2: Dimension options for controls 325
6.3: Order menu commands to move overlapping controls 326
6.4: Table alignment settings for controls 326
6.5: Align menu commands for multiple controls 326
6.6: Pasting text into a control 331
6.7: Properties to hide or disable dialog controls 334
6.8: Quattro Pro areas 335
6.9: Properties common to push buttons, radio buttons, and check boxes 340
6.10: Types of bitmap buttons 342
6.11: Text draw options 343
6.12: Restricting information in an edit field 344
6.13: Additional edit field properties . . . 344
6.14: Restricting spin control values 345
6.15: Additional spin control properties . 345
6.16: Rectangle Style property options . . 346
6.17: List box display properties 347
6.18: Scroll bar Parameter options (the Parameters property) 350

6.19: Events that can initiate a link command 354
6.20: Items in the object pick list 357
6.21: Dialog box/control actions 361
7.1: Menu terms 372
7.2: Events a menu command can detect .376
A.1: Command equivalents by command 383
B.1: Identifying an object 400
B.2: Manipulating a block property 401
B.3: Manipulating a control property . . . 401
B.4: The Color properties 403
B.5: The Dimension property 403
B.6: The Font properties 404
B.7: Common graph object properties . . 404
B.8: Identifying active objects 406
B.9: Identifying menu commands 406
B.10: Active objects and menu commands 407
B.11: Dialog objects and properties 409
B.12: Graph objects (drawn) 418
B.13: Fixed object names 420
B.14: Graph objects (fixed) 421
B.15: Notebook objects 426
B.16: Graphs page icons 429
C.1: ANSI character set 432

F I G U R E S

3.1: A macro flow chart 118
3.2: A commented macro 119
3.3: An interactive macro 124
3.4: A flow macro 125
3.5: Using a subroutine macro 126
3.6: Passing arguments to a subroutine . 127
3.7: A cell macro at work 128
3.8: A file macro 128
3.9: Command equivalents in action 129
3.10: A form input macro 131
3.11: A macro to talk to ObjectVision ... 132
3.12: Changing the menu bar 134
3.13: Creating macro buttons from labels 134
3.14: A macro that updates itself 138
3.15: The macro debug window 139
3.16: An example macro loop 142
4.1: How {ABS} affects parts of a cell address 158
4.2: A DDE conversation list 214
4.3: {Parse.*Option*} command statements with data 237
4.4: A block of formulas (left) and their results (right) 251
4.5: Using {RECALCCOL} 252
4.6: Using {RECALC} 252
5.1: The opening screen in the Budgeteer 284
5.2: Searching for specific records 287
5.3: The Budgeteer SpeedBar 288
5.4: A dialog box in the Budgeteer 292
5.5: A macro that uses {DODIALOG} to display a dialog box 293
5.6: The Budgeteer menu definition 295
5.7: The Budgeteer's autoload and cleanup macros 296
6.1: A sample dialog box 302
6.2: A sample SpeedBar (below the standard SpeedBar) 302
6.3: A new dialog window 303
6.4: A dialog box as the user sees it (left) and as the developer creates it (right) ... 303
6.5: The dialog window SpeedBar 304
6.6: The finished LOANDB dialog box .. 308
6.7: The LOANDB dialog box after adding an edit field and a spin control 309
6.8: The LOANDB dialog box after adding the group box 310
6.9: The group box after adding two radio buttons 310
6.10: The LOANDB dialog box after adding labels 311
6.11: The arguments in the {DODIALOG} command 313
6.12: The notebook after adding the {DODIALOG} command 314
6.13: The Object Link dialog box 315
6.14: The Object Link dialog box after specifying the link command 316
6.15: A dialog window for creating SpeedBars 318
6.16: The QUIKSAVE SpeedBar after adding a push button 319
6.17: The QUIKSAVE SpeedBar after adding a bitmap button 319
6.18: A link command that will save the file 320
6.19: A link command that will draw a box around the active block 321
6.20: The QUIKSAVE SpeedBar as it appears in the notebook 321
6.21: The dialog box with three controls . 328

6.22: The dialog box after reversing the order of the controls 328
6.23: A macro to calculate loan payments 329
6.24: How dragging a handle of a parent control affects the child control 337
6.25: How resizing a parent affects child controls 337
6.26: How child controls move when an edge is unlocked 338
6.27: The Object Links dialog box 353
6.28: The Object Link dialog box after specifying CancelExit as the event . 356
6.29: The Object Link dialog box after specifying DOMACRO as the action 356
6.30: The Object Link dialog box after specifying the macro 356
6.31: A link command that will change the zoom factor of the notebook 358
6.32: A link command that will set the notebook's Zoom factor to 100% ... 360
6.33: A link command that will read a cell's alignment 361
6.34: A link command that will display another dialog box 362
6.35: A link command that will simulate an event 363
6.36: A link command that will track the typeface of the active block 365
6.37: A link command that will update an edit field at specific time intervals . 365
6.38: The dialog window after adding a file control and an edit field 367
7.1: A new File | Save menu command .. 372
7.2: A menu block and macro 373
7.3: Names in a menu block 375
7.4: Link commands in a menu block ... 376
7.5: Adding hint text to menu commands 377
7.6: Adding shortcut keys to a menu block 378
7.7: Specifying when the command is available 379
7.8: Adding divider lines 380

INTRODUCTION

Building Spreadsheet Applications contains information on using Quattro Pro @functions and macros to build spreadsheet applications; it assumes a general knowledge of Quattro Pro procedures and concepts. Once you're acquainted with Quattro Pro, this will probably be the book you refer to the most.

This book contains seven chapters and three appendixes:

- **Chapter 1, "Using @functions,"** provides an overview of how to use @functions.
- **Chapter 2, "@Function reference,"** gives a detailed description of each @function.
- **Chapter 3, "Using macros,"** tells you how to create, edit, and debug macros.
- **Chapter 4, "Macro command reference,"** gives detailed descriptions of all macro commands and command equivalents. A table at the beginning of the chapter summarizes the commands by group.
- **Chapter 5, "Application basics,"** provides an overview of application building in Quattro Pro.
- **Chapter 6, "Dialog boxes and SpeedBars,"** explains how to create and edit dialog boxes and SpeedBars.
- **Chapter 7, "Menus,"** explains how to change the active menu bar, and design custom menus.
- **Appendix A, "Command equivalents,"** lists the macro commands that emulate menu commands and other Quattro Pro operations in macros.
- **Appendix B, "Property reference,"** lists each property that @PROPERTY and property macro commands can manipulate.
- **Appendix C, "ANSI codes,"** is an ANSI table listing decimal and hexadecimal codes for each ANSI character.

C H A P T E R

1

Using @functions

Quattro Pro provides many built-in functions that perform calculations and return values. These functions, called *@functions* because they're preceded by an at-sign (@), are special commands entered in cells, either alone or in formulas. They are grouped into eight categories:

- **Mathematical** @functions are used in mathematical formulas—for example, to determine the trigonometric sine or square root.
- **Statistical** @functions perform analytic calculations on a list of values—for example, to find the average value in a block of cells.
- **Database** @functions perform analytic calculations on records in a database—to find the average value for a field, for example.
- **Logical** @functions are used mostly in conditional statements, where the results are based on the validity of a logical expression, such as `+A3>0`.
- **Financial** @functions primarily calculate investments and cash flow, such as annuity and mortgage payments
- **Date and time** @functions calculate dates and times. For example, @NOW returns the current date or time (depending on the numeric format).
- **String** @functions manipulate character strings or labels—to insert characters in the middle of a string, for example.

- **Miscellaneous** @functions provide current information. For example, @HLOOKUP, @INDEX, and @VLOOKUP return values in a lookup table.

The "@Functions by type" section (page 9) lists @functions within each category.

This chapter describes how to enter and use the different types of @functions to produce a wide range of solutions. Chapter 2 is an alphabetical reference of @functions. Each entry includes syntax, arguments, a short description, and examples.

Entering @functions

It is important to enter @functions in the proper format, or *syntax*:

@FUNCTION(*Argument1*, *Argument2*, ...)

See page 7 for a table of argument types.

Argument is a generic term for the information required by an @function. Most @functions need at least one argument. The type of information required depends on the specific @function.

An *@function statement* is the complete command string, including arguments and punctuation. You enter an @function statement in a cell just as you would any other data: Select the cell and type the entry. The @function appears on the input line. When you press *Enter*, the *result* of the @function appears in the cell. If the result is a numeric value, it becomes a value entry. If the result is a string, it becomes a label entry.

@Function features

These features can help you construct @function statements:

- The **@Functions button** appears in the SpeedBar when you start to type. Choose it to display the @functions choice list, from which you can choose an @function to place in the input line.

When viewing this list, you can press F1 for @functions help.

- The **Functions key** (*Alt+F3*) works like the @Functions button to display a list of @functions. Remember, if the Startup option Compatibility Keys is unchecked in the application Object Inspector, you can start typing the name of an @function to display it.
- **@Functions hints** in the status line list arguments and provide positioning information. As you move through an @function in

the input line, the argument at the insertion point changes to uppercase.

- **Block names** make it easier to reference blocks of cells. Once you've assigned a name to a block, you can reference it by name instead of by its coordinates (see Chapter 2 of the *User's Guide*).
- **Pointing** lets you insert cell references in an @function statement by pointing to them, either with the selector or the mouse. Be sure you're in Point mode. Chapter 2 of the *User's Guide* contains details on pointing.

Syntax rules

You can enter @functions in either uppercase or lowercase letters.

An @function statement's syntax must meet these rules:

- There must be a leading @.
- Required arguments must be enclosed in parentheses. (@Functions that don't require arguments don't need parentheses.)
- Multiple arguments must be separated with commas. (You can change the argument separator to a semicolon or period instead of a comma by choosing International in the application Object Inspector™.)
- Multiple arguments must be entered in the specified order.
- There must not be a space between @ and the @function name. There can be spaces between arguments, parentheses, and the @function name, but Quattro Pro deletes them.

If you enter an @function statement incorrectly, Quattro Pro enters Edit mode and displays the message "Syntax error."

Nesting @functions

You can use more than one @function in a single statement. For example, the following @function statement calculates the average of values in block C14..F24, then rounds the average to two decimal places:

```
@ROUND(@AVG(C14..F24),2)
```

This is called *nesting* @functions. You can nest as many @functions as you like in a statement (up to 1024 characters), but it is more difficult to debug complex @function statements. It is better to break @functions into several smaller statements, which you can reference with a final statement. The smaller pieces make it easier

to pinpoint an error, and, because they're available for use with other formulas, they use memory more efficiently.

Comments and @functions

You can include an onscreen comment with any formula, including those containing @functions. Typically, comments serve as memory jogs, asides, or annotations that you don't want to appear in the notebook itself.

To enter a comment, type a semicolon (;) after the @function statement, then type the text. An @function and comment together can be up to 1024 characters.

Here are some examples:

`@DMAX(A2..E10,4,A13..A14);`computes the top sale in July
`@STD(B4..B11);`remember: a std of 0 means all values in the list are equal
`@MOD(@NOW,7);`0=Sat, 1=Sun, 2=Mon, 3=Tues, 4=Wed, 5=Thurs, 6=Fri

Argument types

There are three general types of arguments:

- numeric values
- block values
- string values

Most arguments require one of these types of values. Some accept a combination or a choice of types. For example, @SUM and the other statistical @functions accept a *List*, which can be a block value (block coordinates or block names) in combination with numeric values.

The next table illustrates how to enter the three types of arguments: numeric values, block values, and string values.

Table 1.1
@Function arguments

Argument	Example
Numeric Values	
Actual value	`@INT(254.933)`
Coordinate of a cell that contains a numeric value	`@INT(A5)`
Linked coordinate referencing a cell in another notebook that contains a numeric value	`@INT([C:\BUDGET]A5)`
Block name of a single-cell block that contains a numeric value	`@INT(TOTAL)`
Formula resulting in a numeric value	`@INT(B4*10)`
@Function resulting in a numeric value	`@ABS(@INT(C4))`
Combination of numeric values	`@ABS(@INT(C4)-35+A2)`
List of numeric values	`@MAX(1,5,A3,G9,25)`
Block Values	
Cell block coordinates	`@MAX(A1..B3)`
Coordinate of a single cell	`@MAX(A1)`
Linked coordinate(s) referencing a cell or block of cells in another notebook	`@MAX([C:\BUDGET]A1..A5)`
Block name	`@MAX(JANUARY)`
Combination of block values (for a *list*)	`@MAX(JANUARY,C15..D25,F10)`
String Values	
Actual string (must use double quotes)	`@PROPER("ACME Co.")`
Coordinate of a cell that contains a label	`@PROPER(D13)`
Linked coordinate referencing a cell in another notebook that contains a label	`@PROPER([C:\BUDGET]D13)`
Block name of a single-cell block containing a label	`@PROPER(COMPANY)`
Formula resulting in a string	`@PROPER(+MONTH&"Sales")`
@Function resulting in a string	`@LENGTH(@PROPER("ACME Co."))`

Operators

You can combine @functions with operators to yield complex @function statements. The next sections summarize operator precedence and the use of different operator types.

Operator precedence

The result of a formula depends on the order in which Quattro Pro performs the requested operations. Each operator has a *precedence*, and operations are performed in order of precedence. For example, because multiplication has greater precedence than addition, `5+1*3` results in 8, *not* 18.

Quattro Pro performs operations with equal precedence from left to right.

The next table lists operators allowed in formulas and the precedence assigned to each. The operator with the highest precedence number (7) is performed first.

Table 1.2
Quattro Pro formula operators

Operator	Description	Precedence
&	String combination	1
#AND# #OR#	Logical AND, logical OR	1
#NOT#	Logical NOT	2
= <>	Equal, not equal	3
< >	Less than, greater than	3
<=	Less than or equal	3
>=	Greater than or equal	3
– +	Subtraction, addition	4
* /	Multiplication, division	5
– +	Negative, positive (unary)	6
^	To the power of (exponentiation)	7

You can override the precedence of operators with parentheses. Any operation enclosed by parentheses is given highest priority. For example, this expression is correct without parentheses:

```
+2+4*6-3=23
```

With parentheses, the result changes as shown:

```
+2+4*(6-3)=14
```

Operator use

The next table shows arithmetic operators, which calculate numeric values.

Table 1.3
Arithmetic operators

Operator	Use	Example
–	Negation or subtraction	`@MOD(B4,4)-A1 +A4-@MOD(B4,4)`
+	Addition	`@MOD(-B4,4)+E12`
*	Multiplication	`@ROUND(E13,0)*2`
/	Division	`@ABS(C9/@PAYMT(A1,12,100,0))`
^	Exponentiation	`@LOG(10^@DAVG(A1..A5,C8..C9))`

The & operator combines two strings into one:

Table 1.4
Text operator

Operator	Use	Example
&	Concatenation	`@LENGTH(C9&"total")`

Logical operators are used in statements that evaluate the relationship between values. If a statement is true, Quattro Pro returns 1; if false, 0. When used with strings, logical operators compare the alphabetical order of the characters in two or more strings. For example, `+"Bob"<"Carlos"` is true.

Table 1.5
Logical operators

Operator	Use	Example
<	Less than	`+64000 < @MEMAVAIL`
>	Greater than	`+C9>@SQRT(F12)`
<=	Less than or equal to	`@RAND<=0.4`
>=	Greater than or equal to	`@ROUND(C5,3)>=F12`
< >	Not equal to	`@AVG(A1..A4)<>@AVG(B1..B4)`
=	Equality	`@IF(A1=@SQRT(B4),"verified", "try again")`
#NOT#	Logical negation	`#NOT#(@ISNA(C12))`
#AND#	Logical AND (both must be true to return 1)	`+A1=B2 #AND# @ISERR(C7)`
#OR#	Logical OR (either can be true to return 1)	`@ISNUMBER(D4) #OR# @ISERR(D4)`

See Chapter 2 of the *User's Guide* for more about using each type of operator in formulas.

ERR and NA

When formula syntax and arguments are correct, most @functions and formulas return either a numeric value (including dates) or string value (labels or text). If a value cannot be calculated, ERR is returned. NA is returned when information is not available. Any @function or other formula that references an ERR or NA cell also returns ERR or NA. @ERR and @NA let you enter these values directly into cells. If ERR and NA both apply, ERR takes precedence. For example, `@ERR+@NA` = ERR. For details on @ERR and @NA, see pages 49 and 74.

Functions by type

This section lists @functions in each of the eight categories. Refer to the individual @functions beginning on page 21 for details and examples. Table 1.1 on page 7 lists the ways you can enter numeric, block, and string arguments.

Mathematical @functions

The mathematical @functions perform calculations with numeric values as arguments; they all return a numeric value. You must enter all angles in radians for @COS, @SIN, and @TAN. Accordingly, @ASIN, @ATAN, and @ATAN return all angles in radians. To convert radians to degrees, use @DEGREES; to convert degrees to radians, use @RADIANS (see pages 41 and 85, respectively).

Every mathematical @function except @PI and @RAND requires numeric values as arguments.

Table 1.6
Mathematical @functions

Mathematical @function	Returns
@ABS(*X*)	The absolute value of *X*
@ACOS(*X*)	The arc cosine of *X*
@ASIN(*X*)	The arc sine of *X*
@ATAN(*X*)	The arc tangent of *X* (2 quadrant)
@ATAN2(*X*,*Y*)	The arc tangent of *Y*/*X* (4 quadrant)
@COS(*X*)	The cosine of angle *X*
@DEGREES(*X*)	The number of degrees in *X* radians
@EXP(*X*)	*e* raised to the *X*th power
@INT(*X*)	The integer portion of *X*
@LN(*X*)	The Log base e of *X*
@LOG(*X*)	The Log base 10 of *X*
@MOD(*X*,*Y*)	The remainder of *X*/*Y*
@PI	The value π (3.14159..)
@RADIANS(*X*)	The number of radians in *X* degrees
@RAND	A random number between 0 and 1
@ROUND(*X*,*Num*)	*X* rounded to the number of digits specified with *Num* (up to 15)
@SIN(*X*)	The sine of angle *X*
@SQRT(*X*)	The positive square root of *X*
@TAN(*X*)	The tangent of angle *X*

Statistical @functions

The statistical @functions perform aggregation and counting operations on a group of values expressed as a list of one or more arguments. These arguments can be numeric values or block values.

If you include a blank cell in the list as a single argument, Quattro Pro treats it as 0. However, if you use a block that contains blank

cell(s) as an argument, Quattro Pro ignores the blank cell(s) in its calculations.

Note If any labels exist in a block included in the list of arguments, Quattro Pro treats them as 0.

Table 1.7
Statistical @functions

Statistical @function	Returns
@AVG(*List*)	The average of *List*
@COUNT(*List*)	A value equal to the number of non-blank cells in *List*
@MAX(*List*)	The maximum value in *List*
@MIN(*List*)	The minimum value in *List*
@STD(*List*)	The population standard deviation of all non-blank values in *List*
@STDS(*List*)	The sample standard deviation of all non-blank values in *List*
@SUM(*List*)	The sum of values in *List*
@SUMPRODUCT(*Block1*,*Block2*)	The dot (scalar) product of the vectors corresponding to the blocks
@VAR(*List*)	The population variance of all non-blank values in *List*
@VARS(*List*)	The sample variance of all non-blank values in *List*

Here are the formulas for the statistical @functions. If the *List* contains n values, say $x_1 \ldots x_n$, then

$$
\begin{aligned}
Count &= n \\
Sum &= x_1+\ldots+x_n \\
Avg &= Sum/n \\
T &= (x_1\text{-}Avg)^2+\ldots+(x_n\text{-}Avg)^2 \\
Var &= T/n \\
Vars &= T/(n\text{-}1) \\
Std &= Sqrt(Var) \\
Stds &= Sqrt(Vars)
\end{aligned}
$$

Quattro Pro can compute standard deviation and variance by two different methods, using two different sets of @functions:

- The *sample* statistics: @STDS, @VARS, @DSTDS, @DVARS
- The *population* statistics: @STD, @VAR, @DSTD, @DVAR

The two methods are closely related and give similar results. The sample statistics are preferred in most applications and are more

commonly used, but are not compatible with some other spreadsheet programs.

- The sample statistics divide the sum of components by *n-1* instead of *n*, where *n* is the number of items. This gives a less biased result when a sample of data is chosen at random from a larger population.
- The population statistics are better if the entire set of possible numbers is available, in which case the exact average, variance, and standard deviation can be calculated.

Database @functions

The database @functions perform the same aggregation and counting operations as statistical @functions, but they operate on selected field entries in a database instead of a list.

Table 1.8
Database @functions

Database @function	**Returns**
@DAVG(*Block,Column,Criteria*)	Average values in *Column* of *Block* that meet *Criteria*
@DCOUNT(*Block,Column,Criteria*)	Number of values in *Column* of *Block* that meet *Criteria*
@DMAX(*Block,Column,Criteria*)	Maximum value in *Column* of *Block* that meets *Criteria*
@DMIN(*Block,Column,Criteria*)	Minimum value in *Column* of *Block* that meets *Criteria*
@DSTD(*Block,Column,Criteria*)	The population standard deviation of values in *Column* of *Block* that meets *Criteria*
@DSTDS(*Block,Column,Criteria*)	The sample standard deviation of values in *Column* of *Block* that meets *Criteria*
@DSUM(*Block,Column,Criteria*)	Sum of values in *Column* of *Block* that meets *Criteria*
@DVAR(*Block,Column,Criteria*)	The population variance of values in *Column* of *Block* that meets *Criteria*
@DVARS(*Block,Column,Criteria*)	The sample variance of values in *Column* of *Block* that meets *Criteria*

All database @functions have the same arguments:

Block = the cell block containing the database, including field names
Column = the number of the column containing the field you want to total. The first column in *Block* is 0, second is 1, and so on.
Criteria = a cell block containing search criteria

For details on building criteria tables, see Chapter 13 of the User's Guide.

Block and *Criteria* must be 2-D blocks. *Criteria* must reference a criteria table with at least two rows and one field name. To use all entries for a field, leave the cells beneath that field name blank when building the table.

Logical @functions

The logical @functions are used mostly in conditional statements, where the results are based on the validity of a logical expression, such as +A3>0. A conditional statement returns either the logical value 1 (true) or the logical value 0 (false).

Table 1.9
Logical @functions

Logical @function	**Returns**
@FALSE	The logical value 0
@FILEEXISTS(*FileName*)	1 if the *FileName* exists; otherwise, 0
@IF(*Cond,TrueExpr,FalseExpr*)	*TrueExpr* if *Cond* is true, *FalseExpr* if *Cond* is false
@ISERR(*X*)	1 if *X* is ERR; otherwise, 0
@ISNA(*X*)	1 if *X* is NA; otherwise, 0
@ISNUMBER(*X*)	1 if *X* is a numeric value; otherwise, 0
@ISSTRING(*X*)	1 if *X* is a string; otherwise, 0
@TRUE	The logical value 1

Financial @functions

There are three basic types of financial @functions: annuity, cash flow, and depreciation. They are listed together in the table, then described separately in the following sections.

Table 1.10: Financial @functions

Financial @function	Returns
@CTERM(*Rate,Fv,Pv*)	The number of compounding periods (kept for backward compatibility—see @NPER)
@DDB(*Cost,Salvage,Life,Period*)	Double-declining depreciation allowance
@FV(*Pmt,Rate,Nper*)	The future value of an annuity (kept for backward compatibility—see @FVAL)
@FVAL(*Rate,Nper,Pmt,<Pv>,<Type>*)	The future value of an annuity (an improved version of the equivalent @FV)
@IPAYMT(*Rate,Per,Nper,Pv,<Fv>,<Type>*)	The portion of a payment amount for a loan that is interest
@IRATE(*Nper,Pmt,Pv,<Fv>,<Type>*)	The periodic interest rate (an improved version of the equivalent @RATE)
@IRR(*Guess,Block*)	The internal rate of return
@NPER(*Rate,Pmt,Pv,<Fv>,<Type>*)	The number of periods (an improved version of the equivalent @CTERM and @TERM)
@NPV(*Rate,Block,<Type>*)	The present value of future cash flow
@PAYMT(*Rate,Nper,Pv,<Fv>,<Type>*)	The payment amount for a loan (an improved version of the equivalent @PMT)
@PMT(*Pv,Rate,Nper*)	The payment amount for a loan (kept for backward compatibility—see @PAYMT)
@PPAYMT(*Rate,Per,Nper,Pv,<Fv>,<Type>*)	The portion of a payment amount for a loan that is principal
@PV(*Pmt,Rate,Nper*)	The present value of an annuity (kept for backward compatibility—see @PVAL)
@PVAL(*Rate,Nper,Pmt,<Fv>,<Type>*)	The present value of an annuity (an improved version of the equivalent @PV)
@RATE(*Fv,Pv,Nper*)	The periodic interest rate (kept for backward compatibility—see @IRATE)
@SLN(*Cost,Salvage,Life*)	Straight-line depreciation allowance
@SYD(*Cost,Salvage,Life,Period*)	Sum-of-the-years'-digits depreciation allowance of an asset
@TERM(*Pmt,Rate,Fv*)	The number of payment periods of an investment (kept for backward compatibility—see @NPER)

Annuity @functions

Most of the financial @functions are used in calculating annuities, which are interest-bearing transactions with cash flow in one direction: out, for example, when paying off loans; or in, for example, when accruing savings interest. The arguments for these annuity @functions are as follows:

Be sure to express Rate and Nper in the same type of time period—for example, if one is months, the other can't be years.

Rate = Interest rate (should be greater than –1).

Nper = Number of periods of the loan or investment (should be an integer greater than 0).

Per = A specific loan or investment period, 1 through *Nper*.

Pmt = Payment.
Pv = Present value.
Fv = Future value.
Type = 0 if payments are at the end of each period, 1 if at the beginning. This optional argument lets you use financial @functions to compute either an *ordinary annuity*, where periodic payments are made at the end of each period, or an *annuity due*, where payments are made at the beginning of each period. Quattro Pro assumes that *Type* = 0 unless you indicate otherwise.

Annuity transactions take place over *Nper* time periods, assumed to be months for simplicity in this description. Some, for example @PPAYMT, calculate for a single period, *Per*. The first transaction assumes *Pv* dollars in addition to *Pmt*. In the case of a loan, *Pv* is the principal.

Each period, you get *Pmt* dollars. On the last transaction, you get *Fv* dollars. Depending on *Type*, the payment of *Pmt* dollars is either at the beginning of each month or at the end. One or two of these numbers, *Pv*, *Pmt*, and *Fv*, must be negative, meaning that you will actually pay money rather than receive it.

Under these conditions, the transaction can be interpreted as a loan, with interest compounded monthly. The five variables are related by this formula:

This formula shows how the annuity variables are related; Quattro Pro uses a more complicated formula to solve for any one of them.

$$Pv(1+Rate)^{Nper}+Pmt\left(\frac{(1+RateType)\,(1+Rate)^{Nper-1}}{Rate}\right)+Fv=0$$

Any one variable can be determined from the other four, much as with a financial calculator. The following table summarizes the relationships among them.

Table 1.11
Relationships among annuity @functions

@Function	**Solves for**	**Arguments**
@CTERM	*Nper*	*Rate,Fv,Pv*
@TERM	*Nper*	*Pmt,Rate,Fv*
@NPER	*Nper*	*Rate,Pmt,Pv,<Fv>,<Type>*
@IRATE	*Rate*	*Nper,Pmt,Pv,<Fv>,<Type>*
@RATE	*Rate*	*Fv,Pv,Nper*
@IPAYMT	interest amount for *Per*th period	*Rate,Per,Nper,Pv,<Fv>,<Type>*
@PPAYMT	principal amount for *Per*th period	*Rate,Per,Nper,Pv,<Fv>,<Type>*
@PAYMT	*Pmt*	*Rate,Nper,Pv,<Fv>,<Type>*
@PMT	*Pmt*	*Pv,Rate,Nper*
@PV	*Pv*	*Pmt,Rate,Nper*

Table 1.11: Relationships among annuity @functions (continued)

@Function	Solves for	Arguments
@PVAL	*Pv*	*Rate,Nper,Pmt,<Fv>,<Type>*
@FV	*Fv*	*Pmt,Rate,Nper*
@FVAL	*Fv*	*Rate,Nper,Pmt,<Pv>,<Type>*

Note In all these @functions, as well as @NPV and @IRR, amounts with positive signs represent money received, and amounts with negative signs represent money paid. This convention applies to arguments and to the results of the @functions. In the obsolete 1-2-3–compatible @functions, @PV, @PMT, @FV, @RATE, @TERM, and @CTERM, the amounts are usually all positive regardless of which way the money changes hands.

***Caution:** Results based on non-integer values for Nper are contrary to the Truth in Lending Law and the rules in the Securities Industries Association Handbook.*

Non-integer values are allowed for *Nper*, and the @functions give results that are consistent with other spreadsheet programs, but which are actually not very meaningful. If you borrow money from a bank for, say, 15.2 months with interest paid monthly, giving *Nper* a value of 15.2 in the financial @functions will only be a rough indicator of what the bank will tell you to pay. In order to compute the figures the way the bank would, you have to consider two transactions, one for 15 months and one for 0.2 months.

Cash flow @functions

@IRR and @NPV are similar to annuity @functions; they assume value changes based on *Rate*. These @functions differ from annuity @functions because they operate on tables of data that record income and expenditures. @IRR calculates the internal rate of change, while @NPV calculates the current value of cash flow values given a set interest rate. Pages 62 and 76 describe details of each @function, with examples.

Depreciation @functions

@DDB, @SLN, and @SYD all calculate reduction in asset value over time. They use these methods, respectively: double-declining balance, straight-line, sum-of-the-years, and variable-rate declining balance.

All three depreciation @functions have these arguments:

Cost = the amount paid for an asset
Salvage = the worth of an asset at the end of its useful life
Life = the expected useful life of an asset

@DDB and @SYD also have:

Period = time period for which you want to determine the depreciation expense

Choose a method depending on the type of asset and your accounting practices. Pages 39, 92, and 96 describe each method and give examples of their use.

Date and time @functions

The date and time @functions calculate dates and times.

For calculation purposes, Quattro Pro stores all dates as serial integers beginning with 0 for December 30, 1899. The minimum, –109,571, equals January 1, 1600; the maximum, 474,816, equals December 31, 3199.

The way a date appears onscreen is a matter of formatting. See Chapter 2 of the *User's Guide* for details on formatting dates.

Quattro Pro stores each expression of time as a decimal fraction where 0.000 represents 00:00:00, and 0.99999 represents 23:59:59. To format time expressions in your notebook, choose Numeric Format in the block Object Inspector.

For example, @TODAY returns the current date as an integer, and @NOW returns the current date with the current time added as a fraction.

Table 1.12
Date and time @functions

Date and time @function	Returns
@DATE(*Yr,Mo,Day*)	A serial date number
@DATEVALUE(*DateString*)	A serial date number
@DAY(*DateTimeNumber*)	The day of the month (1-31)
@HOUR(*DateTimeNumber*)	The hour of the day (0-23)
@MINUTE(*DateTimeNumber*)	The minute of the hour (0-59)
@MONTH(*DateTimeNumber*)	A month number (1-12)
@NOW	The current serial date/time number
@SECOND(*DateTimeNumber*)	A second number (0-59)
@TIME(*Hr,Min,Sec*)	A serial time number
@TIMEVALUE(*TimeString*)	A serial time number
@TODAY	The current serial date number
@YEAR(*DateTimeNumber*)	A year number (-300 to 1299; 0=1900)

String @functions

The string @functions manipulate character strings, or labels. As long as a string in a cell begins with a label-prefix character, Quattro Pro accepts any combination of letters, numbers, and other characters. Strings are entered directly as @function arguments, and must be enclosed in quotes.

When you calculate the position of a character in a string, use 0 for the first position, starting at the left.

Table 1.13: String @functions

String @function	Returns
@CHAR(*Code*)	The character for ANSI decimal code number *Code*
@CLEAN(*String*)	Removes nonprintable ANSI characters from *String*
@CODE(*String*)	The ANSI decimal code for the first character in *String*
@EXACT(*String1*,*String2*)	1 if *String1* and *String2* are identical; otherwise, 0
@FIND(*SubString*,*String*,*StartNum*)	The character position of the first *SubString* found in *String*
@HEXTONUM(*String*)	The decimal value of hex string *String*
@LEFT(*String*,*Num*)	The first *Num* characters in *String*
@LENGTH(*String*)	The number of characters in *String*
@LOWER(*String*)	The lowercase value of *String*
@MID(*String*,*StartNum*,*Num*)	*Num* characters of *String*, beginning with the *StartNum* character position
@N(*Block*)	The numeric value of the upper left cell in *Block* (0 if it's a label)
@NUMTOHEX(*X*)	The hexadecimal value of *X*
@PROPER(*String*)	The text in *String* with the first letter in each word capitalized
@REPEAT(*String*,*Num*)	*String*, repeated *Num* times
@REPLACE(*String*,*StartNum*, *Num*,*NewString*)	Removes *Num* characters from *String*, beginning with *StartNum*, then inserts *NewString* in its place
@RIGHT(*String*,*Num*)	The last *Num* characters in *String*
@S(*Block*)	The string value of the upper left cell in *Block* (blank string if it's a value entry)
@STRING(*X*,*Num*)	Numeric value of *X* as a string, with *Num* decimal places
@TRIM(*String*)	*String* without leading, trailing, or consecutive spaces
@UPPER(*String*)	*String* in uppercase characters
@VALUE(*String*)	Numeric value of *String*

Miscellaneous @functions

The miscellaneous @functions perform a variety of calculations. Refer to the individual entries in the alphabetical reference beginning on page 21.

Table 1.14: Miscellaneous @functions

Miscellaneous @function	Returns
@@(*Cell*)	The contents of the cell specified by cell address label in *Cell*
@CELL(*Attribute,Block*)	The requested attribute of *Block*
@CELLINDEX(*Attribute,Block, Column,Row,<Page>*)	The requested attribute of the cell in the offset position of *Block*
@CELLPOINTER(*Attribute*)	The requested attribute of the current cell
@CHOOSE(*Number,List*)	The value in *List* in the position of *Number*
@COLS(*Block*)	The number of columns in *Block*
@COMMAND(*CommandEquivalent*)	The current value of the given Quattro Pro for Windows command equivalent
@CURVALUE(*GeneralAction, SpecificAction*)	The current value of the given Quattro Pro for DOS menu equivalent
@DDELINK([*AppName* | *Topic*] "*DataToReceive*",*DestBlock*,<*nCols*>, <*nRows*>,<*nSheets*>)	The requested data from the specified application and DDE topic, retrieved to *DestBlock*; optional arguments limit the size of *DestBlock*
@ERR	The value ERR (error)
@HLOOKUP(*X,Block,Row*)	The contents of the cell *Row* number of rows beneath *X* in *Block*
@INDEX(*Block,Column,Row, <Page>*)	The contents of the cell located at the specified column and row in *Block*
@MEMAVAIL	The amount of conventional memory currently available
@MEMEMSAVAIL	The amount of expanded (EMS) memory currently available
@NA	The value of NA (Not Available)
@PROPERTY(*ObjectName.Property*)	The current value of the given property
@ROWS(*Block*)	The number of rows in *Block*
@SHEETS(*Block*)	The number of pages in *Block*
@VERSION	The version number of Quattro Pro
@VLOOKUP(*X,Block,Column*)	The contents of the cell *Column* number of columns to the right of *X* in *Block*

Add-in @functions

Quattro Pro for Windows can access special add-in @functions included in dynamic-link library (DLL) programs. They must be identified appropriately within the DLL.

To reference these @functions in a notebook, use this syntax:

```
@dllname.functionname(functionarguments)
```

For example, this statement calls the @function MEDIAN, included in DLL Stats, with a five-item list as an argument:

```
@Stats.MEDIAN(2,4,6,8,10)
```

You can also call DLL @functions with macros. For details, see page 216.

For information on writing add-in @functions for Quattro Pro, contact Borland Technical Support; the *User's Guide* Introduction lists telephone numbers and addresses.

C H A P T E R

2

@Functions reference

This chapter lists the @functions in alphabetical order. Each entry includes syntax, arguments, and a short description.

@@

Format @@(*Cell*)

Cell = a single cell address or a block name for a single-cell block

@@ is used to reference a cell that contains another cell address or block name that is written as a label. @@ translates the label into a cell or single-cell block reference and returns the contents of that cell. @@ does not accept a block name for a block that is not a single-cell block.

Examples `@@("A15")` = the contents of A15

`@@("BLOCK_NAME")` = the contents of the single-cell block named BLOCK_NAME

`@@(A3)` = 50 if A3 contains the label 'A1 and cell A1 contains the value 50

`@@(A3)` = The label 'Total if A3 contains the label 'Block, which is the name of cell C9, which contains the label 'Total

`@@(A1)` = 0, where A1 contains the label 'B1..B5

`@@("A1")` = B1..B5, where A1 contains the label 'B1..B5
`@@("[NOTEBK1]A1")` = B1..B5, where A1 in the current page of NOTEBK1 contains the label 'B1..B5 and NOTEBK1 is open
`@SUM(@@(A1))` = @SUM(B1..B5), where A1 contains the label 'B1..B5, because Quattro Pro translates the label into cell coordinates for a non-single cell block
`@@("B1..B5")` = ERR (not a single-cell block)
`@SUM(@@("B1..B5"))` = 5 if cells B1 through B5 each contain 1

@ABS

Format @ABS(*X*)

X = a numeric value

@ABS returns the absolute (positive) value of *X*.

Examples
`@ABS(-100)` = 100
`@ABS(100)` = 100
`@ABS(0)` = 0

@ACOS

Format @ACOS(*X*)

X = a numeric value between –1 and 1

@ACOS returns the arc cosine of *X*. The result is the angle (in radians) whose cosine is *X*. To convert radians to degrees, use @DEGREES (page 41).

Examples
`@ACOS(1)` = 0
`@ACOS(0.5)` = 1.047198
`@DEGREES(@ACOS(0.5))` = 60
`@ACOS(@ABS(B10))` = the cosine of the absolute value of B10
`@ACOS(2)` = ERR (means that *X* is greater than 1)

@ASIN

Format @ASIN(*X*)

X = a numeric value between –1 and 1

@ASIN calculates the arc sine of *X*. The result is the angle (in radians) whose sine is *X*. To convert radians to degrees, use @DEGREES (page 41).

Examples
`@ASIN(1)` = 1.570796
`@ASIN(0.25)` = 0.25268
`@DEGREES(@ASIN(0.5))` = 30
`@ASIN(-2)` = ERR (*X* is less than –1)

@ATAN

Format @ATAN(*X*)

X = a numeric value

@ATAN calculates the arc tangent of *X*. The result is the angle (in radians) whose tangent is *X*. To convert radians to degrees, use @DEGREES (page 41).

Examples
`@ATAN(0.5)` = 0.463648
`@ATAN(1)` = 0.785398
`@DEGREES(@ATAN(1))` = 45

@ATAN2

Format @ATAN2(*X*,*Y*)

X = a numeric value
Y = a numeric value

@ATAN2 calculates the arc tangent of the angle represented by the point with (x,y) coordinates *X* and *Y*. The result is the angle (in radians) whose tangent is Y/X. The result is between $-\pi$ and π, with the quadrant chosen appropriately according to the sign of the result, as shown in the next figure.

@ATAN2 and @DEGREES results with corresponding X and Y values

	A	B	C	D	E	F
1	X	Y	ATAN2	DEGREES		
2	0	-5	-1.5708	-90		
3	0	5	1.5708	90		
4	-5	0.00000001	3.1416	180		
5	-5	-0.00000001	-3.1416	-180		
6	-5	0	3.1416	180		
7	5	0	0.0000	0		
8	5	0.00000001	0.00000000	0.00000011		
9	5	-0.00000001	-0.00000000	-0.00000011		
10	1	1	0.7854	45		
11	-1	1	2.3562	135		
12	1	-1	-0.7854	-45		
13	-1	-1	-2.3562	-135		
14						

If both *X* and *Y* are 0, the result is ERR.

Caution! The order of arguments is the same as for 1-2-3, but opposite that of the ATAN2 function in FORTRAN and other programming languages.

To convert radians to degrees, use @DEGREES (page 41).

Examples `@ATAN2(1,2)` = 1.107149
`@DEGREES(@ATAN2(1,1))` = 45

@AVG

Format @AVG(*List*)

List = one or more numeric or block values

@AVG calculates the arithmetic mean of all values in *List*, using the formula:

$$\frac{\Sigma \texttt{List}}{\text{N}}$$

If *List* contains more than one item, they must be separated by commas. If any of the referenced cells contains ERR, the resulting value is ERR.

Caution! @AVG ignores blank cells in a block when it makes its calculations, as @COUNT does. Cells containing blank labels, however, are treated as 0; make sure blank cells are truly empty.

Examples `@AVG(5,20,10,5)` = 10
`@AVG(A..C:A1..A3)` = 5 with these values—page A: A1=1, A2=2, A3=3; page B: A1=4, A2=5, A3=6; page C: A1=7, A2=8, A3=9

The following examples refer to the next figure.

`@AVG(B3..E3)` = $704.50
`@AVG(225,B10..E10)` = $1,983.80
`@AVG(B3..B8,PART)` = $399.00 when D3..D8 is named PART

Monthly sales report

	A	B	C	D	E	F	G
1							
2		JANUARY	FEBRUARY	MARCH	APRIL		
3	JA	$652	$833	$599	$734		
4	MH	$456	$305	$522	$478		
5	RB	$68	$59	$73	$79		
6	PD	$379	$379	$379	$379		
7	AH	$80	$80	$80	$80		
8	MS	$750	$750	$750	$750		
9							
10	TOTAL	$2,385	$2,406	$2,403	$2,500		
11							

@CELL

Format @CELL(*Attribute*,*Block*)

Attribute = any one of the attributes listed below
Block = a block reference or name

Caution! @CELL, @CELLINDEX, and @CELLPOINTER do not recalculate automatically as many other @functions do. Press *F9* to obtain the current value.

@CELL returns the requested attribute of the upper left cell in *Block*.

If you type in or point to a single-cell address when entering *Block*, Quattro Pro converts it to a block reference.

You can enter attributes in either uppercase or lowercase, but you must surround them with double quotes. You can also reference a cell containing an attribute.

These attributes are allowed:

"address"	Address of the upper left cell in *Block*.
"row"	Row number of the upper left cell in *Block* (1 to 8192).
"col"	Column number of the upper left cell in *Block* (1 to 256, corresponding to columns A through IV).
"sheet"	Page number of the upper left cell in *Block* (1 to 256, corresponding to notebook pages A through IV).
"NotebookName"	Referenced notebook name, 8 characters or less.
"NotebookPath"	Full pathname of the referenced notebook.

"TwoDAddress"	2-D address of the referenced cell—`$G$23`, for example. The page name is never returned, even if the referenced cell is on another page or in another notebook.
"ThreeDAddress"	3-D address of the referenced cell—`$A:$G$23`, for example. The page name is always returned.
"FullAddress"	Full address of the referenced cell—`[NOTEBK1]$A:$G$23`, for example. The notebook and page names are always returned.
"contents"	Contents of the upper left cell in *Block*.
"type"	Type of data in the upper left cell in *Block*: **b** if cell is blank **v** if cell contains a number or any formula **l** if cell contains a label
"prefix"	Label-prefix character of the upper left cell in *Block*: ' if label is left-aligned ^ if label is centered " if label is right-aligned \ if label is repeating
"protect"	Protected status of the upper left cell in *Block*: **0** if cell is not protected **1** if cell is protected
"width"	Width of the column containing the upper left cell in *Block* (between 1 and 254).
"rwidth"	Width of *Block*.
"format"	Current numeric format of the upper left cell in *Block*: **Fn** Fixed (n = 0-15) **Sn** Scientific (n = 0-15) **Cn** Currency (n = 0-15) **,n** Commas used to separate thousands (n = 0-15) **G** General **+** +/- (bar graph format) **Pn** Percent (n = 0-15) **D1-D5** is Date D1 = DD-MMM-YY D2 = DD-MMM D3 = MMM-YY D4 = MM/DD/YY, DD/MM/YY, DD.MM.YY, YY-MM-DD D5 = MM/DD, DD/MM, DD.MM, MM-DD **D6-D9** is Time

D6 = HH:MM:SS AM/PM
D7 = HH:MM AM/PM
D8 = HH:MM:SS-24hr, HH.MM.SS-24hr, HH,MM,SS-24hr, HHhMMmSSs
D9 = HH:MM-24hr, HH.MM-24hr, HH,MM, HHhMMm

T Show Formulas (Text)
H Hidden
U User-defined

Examples The examples refer to cells in the next figure.

@CELL("prefix",A3) = '
@CELL("format",B5) = C0
@CELL("type",D4) = v
@CELL("address",A3) = $A$3
@CELL("sheet",A:A6) = 1

Monthly expenses

	A	B	C	D	E	F	G
1							
2		JANUARY	FEBRUARY	MARCH	APRIL		
3	Advertising	$652	$833	$599	$734		
4	Car	$456	$305	$522	$478		
5	Cleaning	$80	$80	$80	$80		
6							

@CELLINDEX

Format @CELLINDEX(*Attribute,Block,Col,Row,<Page>*)

Same as @CELL, but returns the requested attribute of the cell in the specified column and row of *Block* on optional *Page*. The upper left corner of *Block* is column 0, row 0.

@CELLPOINTER

Format @CELLPOINTER(*Attribute*)

Attribute = one or more of the possible attributes

@CELLPOINTER is similar to @CELL in that it returns the requested attribute of a cell. The only difference is that it reads the cell containing the selector. You cannot specify another cell. However, if you move the selector to a different cell then press *F9*, the results of the @CELLPOINTER formula are updated.

The attributes available are the same as those used with @CELL (page 25). You can enter attribute names in either uppercase or lowercase, but each must be enclosed by double quotes.

This @function is useful in macros and @IF statements for quickly determining certain characteristics about the current cell, such as whether there is a label or a value currently in it. For example, the function statement

@IF(@CELLPOINTER("type")="v", "value","label")

tells Quattro Pro to write *value* in the cell if the current cell is a value; otherwise, it writes *label*.

Examples These examples refer to cell A1, which contains the date value 11/19/93.

`@CELLPOINTER("address")` = $A$1
`@CELLPOINTER("col")` = 1
`@CELLPOINTER("contents")` = 34292
`@CELLPOINTER("Format")` = D4
`@CELLPOINTER("type")` = v

@CHAR

Format @CHAR(*Code*)

Code = a numeric value between 1 and 255

@CHAR returns the onscreen character corresponding to the given code. This is useful in generating symbols not on the keyboard.

Refer to Appendix C or any standard ANSI table for the code corresponding to each character.

@CHAR can be used to set up an ANSI table in your notebook. Fill a column of cells with values from 1 to 255, using Block | Fill. In the cell to the right of the first number, use @CHAR to show the screen character for 1, for example, `@CHAR(A1)`. Then copy the formula down the column for the next 255 cells. The copied formulas will display the screen character for each number. (The first 128 will be the same characters as in the ASCII character set.)

Examples `@CHAR(33)` = !
`@CHAR(34)` = "
`@CHAR(35)` = #
`@CHAR(36)` = $

@CHOOSE

Format @CHOOSE(*Number*,*List*)

Number = a positive integer ≤ the number of items in *List*-1
List = a group of numeric or string values separated by commas

@CHOOSE selects and enters a value from the supplied *List*. The value it chooses depends on the value of *Number*. 0 chooses the first value in *List*; 1 chooses the second; 2 chooses the third, and so on. If you specify a cell address for the Number argument, Quattro Pro uses the number contained in the cell. If the cell is blank, the first value is chosen.

List values can be cell addresses, strings, numbers, or a mixture of the three. The total number of characters entered cannot exceed 1024.

@CHOOSE operates on integers only. If you supply a non-integer (such as 1.6433), the decimal values are disregarded.

Examples

`@CHOOSE(0,"Howie","Sarah","Chris ")` = Howie
`@CHOOSE(1,"Howie","Sarah","Chris ")` = Sarah
`@CHOOSE(2,"Howie","Sarah","Chris ")` = Chris
`@CHOOSE(A15,"Howie","Sarah","Chris ")` = Howie, if A15 is 0; Sarah if A15 is 1; Chris if A15 is 2.
`@CHOOSE(3,"Howie","Sarah","Chris ")` = ERR (Number is too large).
`@CHOOSE(@MOD(@NOW,7),"Saturday","Sunday", "Monday","Tuesday","Wednesday ","Thursday", "Friday")` = Wednesday when @NOW has *DateTimeNumber* = 33625.

See also @VLOOKUP (page 103) and @HLOOKUP (page 54), which perform similar tasks.

@CLEAN

Format @CLEAN(*String*)

String = a string value

@CLEAN removes all nonprintable characters (0–31) from a string. It is included here for compatibility with other products.

@CODE

Format @CODE(*String*)

String = a string value

@CODE returns the ANSI code of the first character in *String*. This is the opposite of @CHAR, which returns the character corresponding to the given code.

Examples `@CODE("!")` = 33
`@CODE("Sam")` = 83 (code for S)
`@CODE("#")` = 35
`@CODE("$")` = 36
`@CODE("?")` = 63
`@CODE(hello)` = syntax error (missing quotes)

@COLS

Format @COLS(*Block*)

Block = a block value

@COLS returns the number of columns within the given block.

Examples `@COLS(A1..IV1)` = 256
`@COLS(A1..A1)` = 1
`@COLS(NAME)` = 30 (if the NAME block contains 30 columns)

@COMMAND

Format @COMMAND(*CommandEquivalent*)

CommandEquivalent = a Quattro Pro for Windows command equivalent

@COMMAND returns the current value of a Quattro Pro for Windows command equivalent (see page 129). It is most often used in macros to base the next action on a particular menu setting or to save current settings so they can be restored later.

CommandEquivalent must be enclosed in double quotes. To view a list of acceptable arguments, press *Shift+F3* and choose Command Equivalents, or see Appendix A.

@COMMAND returns strings; even if the setting is a number, it is returned as a string. Not all *CommandEquivalent* entries return a useful value. In general, @COMMAND only returns values for command equivalents that take arguments, usually menu commands that display a current setting or status.

Caution! @COMMAND statements do not recalculate automatically. Press *F9* to obtain the current value.

A related @function which uses Quattro Pro for DOS menu equivalents is @CURVALUE (page 34). Another related @function, @PROPERTY (page 82), returns settings for requested object properties.

Examples

`@COMMAND("Print.Block")` = the currently specified print block
`@COMMAND("Print.Copies")` = the number of copies specified in the Spreadsheet Print dialog box
`@COMMAND("Optimizer.Model_Cell")` = the location where the most recent Optimizer model is stored
`@COMMAND("Application.Display")` = the current Display settings in the application Object Inspector (None, Yes, Yes, Yes, A:A1..B:B2 when Clock Display is set to None, the Display Options are all checked, and the second 3-D Syntax option is checked)

@COS

Format @COS(*X*)

X = a numeric value

@COS returns the cosine of the angle *X*. *X* must be given in radians, not degrees. To convert degrees to radians, use @RADIANS (page 85).

Examples

`@COS(@RADIANS(60))` = 0.5
`@COS(@RADIANS(75))` = 0.258819

`@COS(@RADIANS(45))` = 0.707107
`@COS(@PI/3)` = 0.5

@COUNT

Format @COUNT(*List*)

List = one or more numeric or block values, separated by commas

@COUNT returns the number of non-blank cells in *List*. If more than one block is listed, they must be separated by commas.

Any single cells in *List* are counted as 1, even if they are blank. You can work around this by always using blocks (for example, if A4 is blank, use A3..A4, instead of just A4).

Examples The examples refer to cells in the next figure.

`@COUNT(B3..B8)` = 6
`@COUNT(C1..C11)` = 8
`@COUNT(A11)` = 1
`@COUNT(A11..B11)` = 0
`@COUNT(D3..D10,E1..E4)` = 10
`@COUNT(A1..A5,B11,D3..D11)` = 11

Monthly sales report

	A	B	C	D	E	F	G
1							
2		JANUARY	FEBRUARY	MARCH	APRIL		
3	JA	$652	$833	$599	$734		
4	MH	$456	$305	$522	$478		
5	RB	$68	$59	$73	$79		
6	PD	$379	$379	$379	$379		
7	AH	$80	$80	$80	$80		
8	MS	$750	$750	$750	$750		
9							
10	TOTAL	$2,385	$2,406	$2,403	$2,500		
11							

@CTERM

Format @CTERM(*Rate,Fv,Pv*)

Rate = a numeric value representing the fixed interest rate per compounding period

Fv = a numeric value representing the value the investment will reach at some point

Pv = a numeric value representing the present value of the investment

@CTERM calculates the number of time periods required for an investment of *Pv* to reach a value of *Fv*, while earning interest of *Rate* per compounding period. It uses this formula:

$$\frac{\ln(Fv/Pv)}{\ln(1 + Rate)}$$

An equivalent for this formula using @NPER (see page 75) is

@NPER(*Rate*, 0, –*Pv*, *Fv*)

@CTERM assumes that the investment is an ordinary annuity. @NPER, which is calculated differently than but is related to @CTERM, lets you use an optional argument (*Type*) to indicate whether the investment is an ordinary annuity or an annuity due. "Financial functions" on page 13 describes the relationship between @CTERM and @NPER.

Examples Assuming that your savings account has an annual interest rate of 7%, how long would it take a $3000 deposit to reach $5000? The answer is

`@CTERM(7%,5000,3000)` = 7.55 years

Note If the *Rate* figure is cast in terms of years, the result is in years as well. If the interest were compounded monthly, you'd enter `7%/12` and then divide the answer by 12 to get a monthly figure.

You can also use the more sophisticated Quattro Pro @function @NPER to solve this problem:

`@NPER(7%,0,-3000,5000,0)` = 7.55

Other examples:

`@CTERM(0.07,5000,3000)` = 7.550042
`@CTERM(0.10,5000,3000)` = 5.359612
`@CTERM(0.12,5000,3000)` = 4.50747
`@CTERM(0.12,10000,7000)` = 3.147261

@CURVALUE

Format @CURVALUE(*GeneralAction*,*SpecificAction*)

GeneralAction = a general menu category
SpecificAction = a menu item that requires setting

@CURVALUE returns the current value of a Quattro Pro for DOS menu command setting. It is used in macros, usually to base the next action on a particular menu setting, and is included here for compatibility with Quattro Pro for DOS. To view a list of acceptable arguments, press *Shift+F3* and choose / Commands.

A related @function that uses Quattro Pro for Windows command equivalents is @COMMAND (page 30). Another related @function, @PROPERTY (page 82), returns settings for requested properties.

Both *GeneralAction* and *SpecificAction* must be enclosed in double quotes. They must together create one of the menu-equivalent commands listed in Quattro Pro for DOS *@Function and Macros* manuals.

Not all *GeneralAction*/*SpecificAction* combinations return a useful value. In general, only menu commands that display a current setting or status have menu equivalents that are useful for @CURVALUE. Some settings previously controlled with Quattro Pro for DOS commands are now set through Windows, particularly hardware options and printer settings. These don't return a setting in Quattro Pro for Windows.

Caution! @CURVALUE statements do not recalculate automatically. Press *F9* to obtain the current value.

Examples `@CURVALUE("print","block")` = the currently specified print block
`@CURVALUE("file","save")` = the name of the last file saved

@DATE

Format @DATE(*Yr*,*Mo*,*Day*)

Yr = a numeric value between –300 and 1299
Mo = a numeric value between 1 and 12
Day = a numeric value between 1 and 31

@DATE returns the date/time serial number of the date specified with year, month, and day arguments. This serial number can range from –109,571 (January 1, 1600) to 474,816 (December 31,

3199). December 31, 1899 is 0, so a positive number represents the number of days from December 31, 1899 up to the date referenced in the formula.

Date/time serial numbers are used in notebook calculations. (The fractional portion of a serial number is used for the time @functions.)

To display a date/time serial number in a date format, choose Numeric Format in the block Object Inspector. This suppresses the serial numeric display, showing instead the date in its more common form (for example, Jan-1-94 instead of 34335).

Any illegal dates return ERR as their value, for example, `@DATE(87,2,29)`. (This date corresponds to February 29, 1987, which is impossible; 1987 was not a leap year.)

Examples

`@DATE(93,1,1)` = 33970 (January 1, 1993)
`@DATE(91,9,13)` = 33494 (September 13, 1991)
`@DATE(0,1,1)` = 2 (January 1, 1900)
`@DATE(-300,1,1)` = –109571 (January 1, 1600)
`@DATE(1200,12,31)` = 474816 (December 31, 3199)

@DATEVALUE

Format @DATEVALUE(*DateString*)

DateString = a numeric or string value in any valid date format, enclosed by quotes (or block coordinates or a block name for a block that contains a date string)

@DATEVALUE returns a serial date value that corresponds to the value in *DateString*. If the value in *DateString* is not in the correct format or is not enclosed in quotes, an ERR value or syntax error message is returned. If the *DateString* is entered using the international format, the year, month, and day must be in the same order as the current international date format (set in the application Object Inspector) and the separator character must also agree.

You can display resulting date string values in standard date formats by choosing Numeric Format in the block Object Inspector.

There are five valid formats for *DateString*:

- DD-MMM-YY ("04-Jul-93").

- DD-MMM ("04-Jul") (assumes the current year).
- MMM-YY ("Jul-93") (assumes the first of the month).
- The Long International Date Format specified as the system default, one of which is MM/DD/YY ("07/04/93").
- The Short International Date Format specified as the system default, one of which is MM/DD ("07/18"). This format assumes the current year.

Note The easiest way to enter a date value is with the date keys (*Ctrl+Shift+D*). @DATEVALUE is included for compatibility with other products.

Examples

`@DATEVALUE("07/04/93")` = 34154
`@DATEVALUE("JUL-93")` = 34151
`@DATEVALUE("07/18/93")` = 34168
`@DATEVALUE("04-may-92")` = 33728
`@DATEVALUE(07/04/93)` = .018817 (Quattro Pro divides the numbers when quotation marks are omitted)
`@DATEVALUE("May/1992")` = ERR

@DAVG

Format @DAVG(*Block,Column,Criteria*)

Block	=	the 2-D block containing the database, including field names
Column	=	the number of the column containing the field you want to average (the first column in *Block* is 0, the second is 1, and so on)
Criteria	=	a 2-D block containing search criteria; the first row must be field names

@DAVG averages selected field entries in a database. It includes only those entries in column number *Column* whose records meet the criteria specified in *Criteria*.

Criteria are the coordinates of a block containing a *criteria table* (a table specifying search information). For details on criteria tables, see Chapter 13 of the *User's Guide*.

The field specified in your criteria and the field being averaged need not be the same. The field averaged is the one contained within the column you specify as *Column*.

You can specify all or part of your database as *Block*, but field names *must* be included for each field you include in the block.

Examples These examples refer to the next figure.

`@DAVG(A2..E10,4,A13..A14)` = $21,709 (average of July sales)
`@DAVG(A2..E10,4,B13..B15)` = $17,629 (average of California sales)
`@DAVG(A2..E10,4,C13..C14)` = $19,333 (average of DL's sales)
`@DAVG(A2..E10,4,D13..D14)` = $14,716 (average of E456 sales)
`@DAVG(A2..E10,5,A13..A14)` = ERR (*Column* figure too high)
`@DAVG(A2..E10,2,A13..A14)` = 0 (labels are treated as 0)
`@DAVG(A3..E10,4,A13..A14)` = $19,998 (incorrect—field names not included)

Sales by location

	A	B	C	D	E	F
1						
2	DATE	LOCATION	REP	PRODUCT	AMOUNT	
3	Jul-92	San Fran	CJ	E456	$14,999	
4	Jul-92	San Jose	RX	B922	$25,000	
5	Jul-92	San Jose	DL	C234	$18,998	
6	Jul-92	NY	RX	B922	$27,840	
7	Aug-92	San Fran	CJ	E456	$15,500	
8	Aug-92	Chicago	DL	C234	$19,000	
9	Aug-92	San Jose	CJ	E456	$13,650	
10	Aug-92	NY	DL	C234	$20,000	
11						
12		CRITERIA TABLE				
13	DATE	LOCATION	REP	PRODUCT		
14	Jul-92	San Fran	DL	E456		
15		San Jose				
16						

@DAY

Format @DAY(*DateTimeNumber*)

DateTimeNumber = a number between –109,571 (January 1, 1600) and 474,816.9999999 (December 31, 3199)

Converts the date/time serial number you supply as *DateTimeNumber* into the number associated with that day. Decimal (time) portions of the number are ignored.

Examples `@DAY(33940)` = 2 (12/2/92)
`@DAY(34051)` = 23 (3/23/93)
`@DAY(@DATE(93,9,10))` = 10
`@DAY(600,001)` = ERR because the number you entered was larger than 474816.9999999

@DCOUNT

Format @DCOUNT(*Block,Column,Criteria*)

Block = the 2-D block containing the database, including field names

Column = the number of the column containing the field you want to count (the first column in *Block* is 0, the second is 1, and so on)

Criteria = a 2-D block containing search criteria; the first row must be field names

@DCOUNT counts selected field entries in a database. It includes only those entries in column number *Column* whose records meet the criteria specified in block *Criteria*.

Criteria are the coordinates of a block containing a *criteria table* (a table specifying search information). For details on criteria tables, see Chapter 13 of the *User's Guide*.

The field specified in the criteria table and the field being counted need not be the same. The field counted is the one contained within the column you specify as *Column*.

You can specify all or part of your database as *Block*, but field names *must* be included for each field you include in the block.

Examples These examples refer to the next figure.

`@DCOUNT(A2..E10,4,A13..A14)` = 4 (number of July sales)
`@DCOUNT(A2..E10,4,B13..B15)` = 5 (number of California sales)
`@DCOUNT(A2..E10,4,C13..C14)` = 3 (number of DL's sales)
`@DCOUNT(A2..E10,4,D13..D14)` = 3 (number of E456 sales)
`@DCOUNT(A2..E10,5,A13..A14)` = ERR (*Column* figure too high)
`@DCOUNT(A3..E10,4,A13..A14)` = (incorrect—field names not included)

Sales by location

	A	B	C	D	E	F
1						
2	DATE	LOCATION	REP	PRODUCT	AMOUNT	
3	Jul-92	San Fran	CJ	E456	$14,999	
4	Jul-92	San Jose	RX	B922	$25,000	
5	Jul-92	San Jose	DL	C234	$18,998	
6	Jul-92	NY	RX	B922	$27,840	
7	Aug-92	San Fran	CJ	E456	$15,500	
8	Aug-92	Chicago	DL	C234	$19,000	
9	Aug-92	San Jose	CJ	E456	$13,650	
10	Aug-92	NY	DL	C234	$20,000	
11						
12		CRITERIA TABLE				
13	DATE	LOCATION	REP	PRODUCT		
14	Jul-92	San Fran	DL	E456		
15		San Jose				
16						

@DDB

Format @DDB(*Cost*,*Salvage*,*Life*,*Period*)

Cost = a numeric value representing the amount paid for an asset

Salvage = a numeric value representing the worth of an asset at the end of its useful life

Life = a numeric value representing the expected useful life of an asset

Period = a numeric value representing the time period for which you want to determine the depreciation expense

These statements must be true:

$Life \geq Period \geq 1$
Life and *Period* must be integers
$Cost \geq Salvage \geq 0$

@DDB determines accelerated depreciation values for an asset, given the initial cost, life expectancy, end value, and depreciation period. It calculates depreciation using the double-declining balance method.

Depreciation value (*DDB*) and book value (*BV*) are calculated by:

$BV = Cost$
$DDB = 2BV/Life$
$BV = BV - DDB$

DDB is adjusted, if necessary, to ensure that $DDB \geq 0$, and the subsequent calculation of *BV* has $BV \geq Salvage$.

Examples Suppose you just bought a new $4000 computer. Your dealer says you can sell it back to the store for $350 after eight years, but that no one would want to buy it after that. In other words, the *Salvage* value of that computer is $350 and its *Life* is 8. To calculate what the depreciation allowance of this computer will be by the second year (according to the double-declining balance method), enter this formula:

```
@DDB(4000,350,8,2)
```

The result is $750.

These examples show depreciation values for the first five years of a $15,000 investment with a salvage value of $3000 and a life of 10 years:

`@DDB(15000,3000,10,1)` = $3,000
`@DDB(15000,3000,10,2)` = $2,400
`@DDB(15000,3000,10,3)` = $1,920
`@DDB(15000,3000,10,4)` = $1,536
`@DDB(15000,3000,10,5)` = $1,229

@DDELINK

Format @DDELINK([*AppName* | *Topic*]"*DataToReceive*", <*nCols*>,<*nRows*>,<*nSheets*>)

AppName	=	the DDE-server application to contact
Topic	=	the table, spreadsheet, document, or other file in the DDE-server application from which to retrieve data
DataToReceive	=	the field, block, or other information to receive from the application (DDE Item string)
nCols	=	the number of columns in the data block (optional)
nRows	=	the number of rows in the data block (optional)
nSheets	=	the number of pages in the data block (optional)

@DDELINK creates a "live" data link from another Windows application that supports DDE (Dynamic Data Exchange). Using @DDELINK is equivalent to choosing Edit | Paste Link with data

from another application copied to the Clipboard. (Chapter 11 of the *User's Guide* contains details on creating DDE links.)

@DDELINK sets up a zone of cells that can be overwritten whenever data changes in the DDE-server application. Avoid storing other data near @DDELINK, and consider using the limit arguments.

AppName and *Topic* are the same arguments used in {INITIATE} (page 213), except that `"System"` isn't accepted as a substitute for *Topic*. *DataToReceive* is a string indicating the location of the target data in the server application.

When you enter @DDELINK into a cell, the linked data appears there. Unless you indicate otherwise, the data takes up as much space as it did in the original application. You can use *nCols*, *nRows*, and *nSheets* to specify smaller dimensions. If any of these arguments is 0 or omitted, the original dimension applies.

Examples This formula gets information from the field Task in the ObjectVision application TASKLIST.OVD and displays the data in the active notebook:

```
@DDELINK([VISION|TASKLIST.OVD]"Task")
```

The maximum size of the data block for the following formula is 5 cells by 5 cells:

```
@DDELINK([EXCEL|FILE1]"R1C1:R5C5")
```

With *nRows* = 3, the maximum size of the data block drops to 5 cells by 3 cells:

```
@DDELINK([EXCEL|FILE1]"R1C1:R5C5",0,3)
```

@DEGREES

Format @DEGREES(*X*)

X = a numeric value representing radians

@DEGREES converts the given number of radians to degrees, using this formula:

$$\frac{180}{\pi} X$$

Examples `@DEGREES(0.5)` = 28.64789
`@DEGREES(0.017)` = 0.974028
`@DEGREES(@PI/2)` = 90

@DMAX

Format @DMAX(*Block*,*Column*,*Criteria*)

Block = the 2-D block containing the database, including field names

Column = the number of the column containing the field for which you want to find the maximum value (the first column in *Block* is 0, the second is 1, and so on)

Criteria = a 2-D block containing search criteria; the first row must be field names

@DMAX finds the maximum value of selected field entries in a database. It includes only those entries in column number *Column* whose records meet the criteria specified in block *Criteria*.

Criteria are the coordinates of a block containing a *criteria table* (a table specifying search information). For more details on criteria tables, see Chapter 13 of the *User's Guide*.

The field specified in the criteria table and the field you are finding the maximum value for need not be the same. The field you are finding the maximum value for is the one contained within the column you specify as *Column*.

You can specify all or part of your database as *Block*, but field names *must* be included for each field you include in the block.

Examples These examples refer to the next figure.

`@DMAX(A2..E10,4,A13..A14)` = $27,840 (highest July sale)
`@DMAX(A2..E10,4,B13..B15)` = $25,000 (highest California sale)
`@DMAX(A2..E10,4,C13..C14)` = $20,000 (highest of DL's sales)
`@DMAX(A2..E10,4,D13..D14)` = $15,500 (highest of E456 sales)
`@DMAX(A2..E10,5,A13..A14)` = ERR (*Column* figure too high)
`@DMAX(A3..E10,4,A13..A14)` = 27840 (incorrect—the field names for the block aren't included)

Sales by location

	A	B	C	D	E	F
1						
2	DATE	LOCATION	REP	PRODUCT	AMOUNT	
3	Jul-92	San Fran	CJ	E456	$14,999	
4	Jul-92	San Jose	RX	B922	$25,000	
5	Jul-92	San Jose	DL	C234	$18,998	
6	Jul-92	NY	RX	B922	$27,840	
7	Aug-92	San Fran	CJ	E456	$15,500	
8	Aug-92	Chicago	DL	C234	$19,000	
9	Aug-92	San Jose	CJ	E456	$13,650	
10	Aug-92	NY	DL	C234	$20,000	
11						
12		CRITERIA TABLE				
13	DATE	LOCATION	REP	PRODUCT		
14	Jul-92	San Fran	DL	E456		
15		San Jose				
16						

@DMIN

Format @DMIN(*Block,Column,Criteria*)

Block = the 2-D block containing the database, including field names

Column = the number of the column containing the field you want to find the minimum value for (the first column in *Block* is 0, the second is 1, and so on)

Criteria = a 2-D block containing search criteria; the first row must be field names

@DMIN finds the minimum value of selected field entries in a database. It includes only those entries in column number *Column* whose records meet the criteria specified in block *Criteria*.

Criteria are the coordinates of a block containing a *criteria table* (a table specifying search information). For more details on criteria tables, see Chapter 13 of the *User's Guide*.

The field specified in the criteria table and the field for which you are finding the minimum value need not be the same. The field for which you are finding the minimum value is the one contained within the column you specify as *Column*.

You can specify all or part of your database as *Block*, but field names *must* be included for each field you include in the block.

Examples These examples refer to the next figure.

`@DMIN(A2..E10,4,A13..A14)` = $14,999 (smallest July sale)
`@DMIN(A2..E10,4,B13..B15)` = $13,650 (smallest California sale)
`@DMIN(A2..E10,4,C13..C14)` = $18,998 (smallest of DL's sales)

`@DMIN(A2..E10,4,D13..D14)` = $13,650 (smallest E456 sale)
`@DMIN(A3..E10,4,A13..A14)` = 13,650 (incorrect—the field names for the block aren't included)
`@DMIN(A2..E10,5,A13..A14)` = ERR (*Column* figure too high)
`@DMIN(A2..E10,2,A13..A14)` = 0

Sales by location

	A	B	C	D	E	F
1						
2	DATE	LOCATION	REP	PRODUCT	AMOUNT	
3	Jul-92	San Fran	CJ	E456	$14,999	
4	Jul-92	San Jose	RX	B922	$25,000	
5	Jul-92	San Jose	DL	C234	$18,998	
6	Jul-92	NY	RX	B922	$27,840	
7	Aug-92	San Fran	CJ	E456	$15,500	
8	Aug-92	Chicago	DL	C234	$19,000	
9	Aug-92	San Jose	CJ	E456	$13,650	
10	Aug-92	NY	DL	C234	$20,000	
11						
12		CRITERIA TABLE				
13	DATE	LOCATION	REP	PRODUCT		
14	Jul-92	San Fran	DL	E456		
15		San Jose				
16						

@DSTD

Format @DSTD(*Block*,*Column*,*Criteria*)

Block = the 2-D block containing the database, including field names

Column = the number of the column containing the field for which you want to find the standard deviation (the first column in *Block* is 0, the second is 1, and so on)

Criteria = a 2-D block containing search criteria; the first row must be field names

@DSTD finds the population standard deviation for selected field entries in a database. @DSTDS (page 45) computes the standard deviation of sample data. (See page 11 for an explanation of the difference between population statistics and sample statistics.)

@DSTD includes only those entries in column number *Column* whose records meet the criteria specified in block *Criteria*.

Criteria are the coordinates of a block containing a *criteria table* (a table specifying search information). For more details about criteria tables, see Chapter 13 of the *User's Guide*.

The field specified in the criteria table and the field for which you are finding the standard deviation need not be the same. The field

for which you are finding the standard deviation is the field contained within the column you specify as *Column*.

You can specify all or part of your database as *Block*, but field names *must* be included for each field you include in the block.

Examples These examples refer to the next figure.

`@DSTD(A2..E10,4,A13..A14)` = $5,020 (SD of July sales)
`@DSTD(A2..E10,4,B13..B15)` = $4,086 (SD of California sales)
`@DSTD(A2..E10,4,C13..C14)` = $472 (SD of DL's sales)
`@DSTD(A2..E10,4,D13..D14)` = $781 (SD of E456 sales)
`@DSTD(A3..E10,4,A13..A14)` = $4,614 (incorrect—field names for the block aren't included)
`@DSTD(A2..E10,2,A13..A14)` = 0
`@DSTD(A2..E10,5,A13..A14)` = ERR (*Column* figure too high)

Sales by location

	A	B	C	D	E	F
1						
2	DATE	LOCATION	REP	PRODUCT	AMOUNT	
3	Jul-92	San Fran	CJ	E456	$14,999	
4	Jul-92	San Jose	RX	B922	$25,000	
5	Jul-92	San Jose	DL	C234	$18,998	
6	Jul-92	NY	RX	B922	$27,840	
7	Aug-92	San Fran	CJ	E456	$15,500	
8	Aug-92	Chicago	DL	C234	$19,000	
9	Aug-92	San Jose	CJ	E456	$13,650	
10	Aug-92	NY	DL	C234	$20,000	
11						
12		CRITERIA TABLE				
13	DATE	LOCATION	REP	PRODUCT		
14	Jul-92	San Fran	DL	E456		
15		San Jose				
16						

@DSTDS

Format @DSTDS(*Block*,*Column*,*Criteria*)

Block = the 2-D block containing the database, including field names
Column = the number of the column containing the field for which you want to find the standard deviation (the first column in *Block* is 0, the second is 1, and so on)
Criteria = a 2-D block containing search criteria; the first row must be field names

@DSTDS finds the sample standard deviation for selected field entries in a database. @DSTD (page 44) computes the standard

deviation of population data. (See page 11 for an explanation of the difference between sample statistics and population statistics.)

Refer to the @DSTD entry for details and examples of how to use @DSTDS.

@DSUM

Format @DSUM(*Block,Column,Criteria*)

Block = the 2-D block containing the database, including field names
Column = the number of the column containing the field you want to total (the first column in *Block* is 0, the second is 1, and so on)
Criteria = a 2-D block containing search criteria; the first row must be field names

@DSUM totals selected field entries in a database. It includes only those entries in column number *Column* whose records meet the criteria specified in block *Criteria.*

Criteria are the coordinates of a block containing a *criteria table* (a table specifying search information). For more details on criteria tables, see Chapter 13 of the *User's Guide.*

The field specified in the criteria table and the field you are finding the sum of need not be the same. The field you are finding the sum of is the one contained within the column you specify as *Column.*

You can specify all or part of your database as *Block,* but field names *must* be included for each field you include in the block.

Examples These examples refer to the next figure.

`@DSUM(A2..E10,4,A13..A14)` = $86,837 (total of July sales)
`@DSUM(A2..E10,4,B13..B15)` = $88,147 (total of California sales)
`@DSUM(A2..E10,4,C13..C14)` = $57,998 (total of DL's sales)
`@DSUM(A2..E10,4,D13..D14)` = $44,149 (total of E456 sales)
`@DSUM(A3..E10,4,A13..A14)` = 139988 (incorrect—field names for the block are not included)
`@DSUM(A2..E10,2,A13..A14)` = 0
`@DSUM(A2..E10,5,A13..A14)` = ERR (*Column* figure too high)

Sales by location

	A	B	C	D	E	F
1						
2	DATE	LOCATION	REP	PRODUCT	AMOUNT	
3	Jul-92	San Fran	CJ	E456	$14,999	
4	Jul-92	San Jose	RX	B922	$25,000	
5	Jul-92	San Jose	DL	C234	$18,998	
6	Jul-92	NY	RX	B922	$27,840	
7	Aug-92	San Fran	CJ	E456	$15,500	
8	Aug-92	Chicago	DL	C234	$19,000	
9	Aug-92	San Jose	CJ	E456	$13,650	
10	Aug-92	NY	DL	C234	$20,000	
11						
12		CRITERIA TABLE				
13	DATE	LOCATION	REP	PRODUCT		
14	Jul-92	San Fran	DL	E456		
15		San Jose				
16						

@DVAR

Format @DVAR(*Block,Column,Criteria*)

Block = the 2-D block containing the database, including field names

Column = the number of the column containing the field for which you want to compute variance (the first column in *Block* is 0, the second is 1, and so on)

Criteria = a 2-D block containing search criteria; the first row must be field names

@DVAR calculates the population variance for selected field entries in a database. @DVARS (page 48) computes the variance of sample data. (See page 11 for an explanation of the difference between population statistics and sample statistics.)

@DVAR includes only those entries in column number *Column* whose records meet the criteria specified in block *Criteria*.

Criteria are the coordinates of a block containing a *criteria table* (a table specifying search information). For more details about criteria tables, see Chapter 13 of the *User's Guide*.

The field specified in the criteria table and the field for which you are calculating the variance need not be the same. The field counted is the field contained within the column you specify as *Column*.

You can specify all or part of your database as *Block*, but field names *must* be included for each field you include in the block.

Examples These examples refer to the next figure.

`@DVAR(A2..E10,4,A13..A14)` = 25198366 (variance of July sales)
`@DVAR(A2..E10,4,B13..B15)` = 16697557 (variance of California sales)
`@DVAR(A2..E10,4,C13..C14)` = 222668 (variance of DL's sales)
`@DVAR(A2..E10,4,D13..D14)` = 610367 (variance of E456 sales)
`@DVAR(A3..E10,4,A13..A14)` = 21291726 (incorrect—field names for the block aren't included)
`@DVAR(A2..E10,2,A13..A14)` = 0
`@DVAR(A2..E10,5,A13..A14)` = ERR (*Column* figure too high)

Sales by location

	A	B	C	D	E	F
1						
2	DATE	LOCATION	REP	PRODUCT	AMOUNT	
3	Jul-92	San Fran	CJ	E456	$14,999	
4	Jul-92	San Jose	RX	B922	$25,000	
5	Jul-92	San Jose	DL	C234	$18,998	
6	Jul-92	NY	RX	B922	$27,840	
7	Aug-92	San Fran	CJ	E456	$15,500	
8	Aug-92	Chicago	DL	C234	$19,000	
9	Aug-92	San Jose	CJ	E456	$13,650	
10	Aug-92	NY	DL	C234	$20,000	
11						
12		CRITERIA TABLE				
13	DATE	LOCATION	REP	PRODUCT		
14	Jul-92	San Fran	DL	E456		
15		San Jose				
16						

@DVARS

Format @DVARS(*Block*,*Column*,*Criteria*)

Block = the 2-D block containing the database, including field names

Column = the number of the column containing the field for which you want to compute variance (the first column in *Block* is 0, the second is 1, and so on)

Criteria = a 2-D block containing search criteria; the first row must be field names

@DVARS calculates the sample variance for selected field entries in a database. @DVAR (page 47) computes variance with population data. (See page 11 for an explanation of the difference between population statistics and sample statistics.)

Refer to the @DVAR entry for details and examples of how to use @DVARS.

@ERR

Format @ERR

@ERR returns the value ERR in the current cell and in any other cells that reference the current cell, either directly or indirectly. (Exceptions to this are @COUNT, @DCOUNT, @ISERR, @ISNA, @ISNUMBER, @ISSTRING, and @CELL formulas; these will not result in ERR if they reference an ERR cell.)

The ERR value resulting from this @function is the same as the ERR value produced by Quattro Pro when it encounters an error. It is often used with @IF to bring attention to error conditions.

Note ERR is a unique number, not to be confused with the label ERR.

Examples `@ERR` = ERR
`@IF(B6>B7,0,@ERR)` = 0 (if B6>B7) or ERR (if B6<B7)

@EXACT

Format @EXACT(*String1*,*String2*)

String1 = a valid string value
String2 = a valid string value

@EXACT compares the values of *String1* and *String2*. If the values are *exactly* identical, including capitalization and diacritical marks (such as ~), it returns 1. If there are any differences, it returns 0.

If you're comparing literal strings, surround them with double quotes. If you use a block name, no quotes are necessary. You can compare the contents of label cells only. If you try to compare one or more numbers or empty cells, the result is ERR. When you compare labels, label prefixes are ignored.

To compare strings or cell contents without regard to capitalization or diacritical marks, use @IF. For example, `@IF(C3=B3,1,0)` returns 1 if the contents of the cells are the same but are capitalized differently.

Examples `@EXACT("client","Client")` = 0
`@EXACT("client","client")` = 1
`@EXACT(29,"29")` = ERR (the first string is a value)
`@EXACT(A1,"yes")` = 1 (if A1 contains the label yes)
`@EXACT("client","client","client")` = syntax error
(more than two strings)

@EXP

Format @EXP(*X*)

X = a numeric value ≤ 709

@EXP returns the mathematical constant *e*, raised to the *X*th power. This @function is the inverse of a natural logarithm, @LN (page 68).

Examples
`@EXP(3.4)` = 29.9641000474
`@EXP(1)` = 2.718281828459 (the actual value of *e*)
`@SQRT(@EXP(2))` = 2.71828183
`@LN(@EXP(2.5))` = 2.5

@FALSE

Format @FALSE

@FALSE returns the logical value 0 and is usually used in @IF formulas. The zero that it returns is the same as any other zero, but @FALSE makes the formula easier to read.

See also @TRUE on page 101.

Examples
`@FALSE` = 0
`@IF(C3=100,10,@FALSE)` = 10, if C3 = 100; or 0, if C3 ≠ 100
`@IF(C3=100,@TRUE,@FALSE)` = 1, if C3 = 100; or 0, if C3 ≠ 100

@FILEEXISTS

Format @FILEEXISTS(*FileName*)

FileName = any file name

@FILEEXISTS returns a 1 if a file named *FileName* exists in the current file directory, and returns a 0 if it doesn't. *FileName* can be a block name containing a path or file name string. If entered as a literal string, *FileName* must be enclosed by quotes and must include any file-name extension attached to the file name. To search for a file in a directory other than the default directory, include the directory path in *FileName*.

Examples @FILEEXISTS("EXAMPLE.WB1") = 1 (if EXAMPLE.WB1 is in the current directory)
@FILEEXISTS("C:\DATA\EXAMPLE.WB1") = 1 (if EXAMPLE.WB1 is in the specified directory)
@FILEEXISTS(FILE_NAME) = 1 (if the block FILE_NAME contains a path and file-name label and if the file exists in that directory)
@FILEEXISTS(A1) = 1 (if A1 contains C:\DATA\EXAMPLE.WB1 and EXAMPLE.WB1 is in the specified directory)

@FIND

Format @FIND(*Substring,String,StartNumber*)

SubString	=	a valid string value, representing the value to search for
String	=	a valid string value, representing the value to search through
StartNumber	=	a numeric value ≥ 0, representing the character position to begin searching with; 0 = the first character

@FIND searches through the given *String* from left to right for the value given as *Substring*. If it finds *Substring*, it returns the character position at which the first occurrence was found. *StartNumber* begins the search at that number of characters into the string: 0 = the first character in the string, 1 = the second, and so on. The value of *StartNumber* must not be more than the number of characters in *String* minus 1.

@FIND is case-sensitive and is also sensitive to diacritical marks used in non-English languages. You can overcome the case-sensitivity of this @function by using @UPPER to force one or more of the strings into all caps; for example, this formula forces both the substring in cell C3 and the string in cell C4 to uppercase, then searches for the substring:

```
@FIND(@UPPER(C3),@UPPER(C4),0)
```

@FIND is most often used in conjunction with two other string functions: @REPLACE (to perform "search and replace" operations on strings) and @MID (to access substrings).

If @FIND fails to find any occurrences of *Substring*, or if the *StartNumber* given is invalid, the result is ERR.

Examples

```
@FIND("i","find",0) = 1
@FIND("nd","find",2) = 2
@FIND("F","find",0) = ERR
@FIND("f","find",3) = ERR
@FIND("d","find",4) = ERR
@FIND(n,find,0) = syntax error (quote marks omitted from strings)
@FIND("hi",C4,0) = 1 (if C4 contains ship)
@FIND(A1,A2,0) = 1 (if A1 contains es and A2 contains test)
```

@FV

Format @FV(*Pmt,Rate,Nper*)

Pmt = a numeric value representing the amount of the periodic payment
Rate = a numeric value > –1, representing periodic interest rate
Nper = number of periods, which should be an integer ≥ 2

@FV returns the future value of an investment where *Pmt* is invested for *Nper* periods at the rate of *Rate* per period. @FV calculates the future value with this formula:

$$Pmt \frac{(1 + Rate)^{Nper} - 1}{Rate}$$

An equivalent for this formula using @FVAL (see page 53) is

@FVAL(*Rate, Nper, –Pmt,* 0)

@FV assumes that the investment is an ordinary annuity. @FVAL, which is calculated differently than but is related to @FV, lets you use an optional argument (*Type*) to indicate whether the investment is an ordinary annuity or an annuity due. "Financial functions" on page 13 describes the relationship between @FV and @FVAL.

Examples Assume you want to set aside $500 at the end of each year in a savings account that earns 15% annually. To determine what the account will be worth at the end of six years, enter this formula:

```
@FV(500,15%,6)
```

Your yearly payment of $500 will be worth $4,376.87 in six years.

You could also use @FVAL in this way:

```
@FVAL(15%,6,-500,0,0)
```

In @FVAL, you must be precise about whether a payment is out of your pocket (a negative number) or paid to you (a positive number).

Other examples:

`@FV(200,.12,5)` = $1,270.57
`@FV(500,0.9,4)` = $6,684.50
`@FV(800,0.9,3)` = $5,208.00
`@FV(800,0.9,A3)` = $40,929.67 (if A3 = 6)

@FVAL

Format @FVAL(*Rate*,*Nper*,*Pmt*,<*Pv*>,<*Type*>)

Rate = a numeric value > –1, representing periodic interest rate
Nper = number of periods, which should be an integer > 0
Pmt = a numeric value representing the amount of the periodic payment
Pv = present value
Type = 0 if payments are at the end of each period, 1 if they are at the beginning

Be sure to enter a negative number for money that's out of your pocket and a positive number for money that's coming to you.

@FVAL is another version of the 1-2-3–compatible @function @FV (see the preceding entry). The last two arguments, *Pv* and *Type*, are optional. If you omit the last one or both of them, Quattro Pro assumes their values are zero. See "Financial functions" on page 13 for more information.

Examples For example, assume you want to set aside $500 at the *start* of each year in a savings account that earns 15% annually. To determine what the account will be worth at the end of six years, starting at a present value of zero, enter this formula:

Putting money into an account before it earns its interest for that year means you have an annuity due to you, which increases the future value calculations.

```
@FVAL(15%,6,-500,0,1)
```

Your yearly payment of $500 will be worth $5,033.40 in six years, or $656.53 more than if you deposited the money on the last day of the year as in the previous @FV example.

If the account already had $340 in it before your yearly deposits of $500, you could calculate the future value after six years with this formula:

`@FVAL(15%,6,-500,-340,1)` = $5,819.84

@HEXTONUM

Format @HEXTONUM(*String*)

String = a hexadecimal number enclosed by quotes

@HEXTONUM converts the hexadecimal number in the string to the corresponding decimal value. @NUMTOHEX, described on page 78, performs the opposite conversion, from decimal to hex.

Examples
`@HEXTONUM("a")` = 10
`@HEXTONUM("10")` = 16
`@HEXTONUM("00FF")` = 255
`@HEXTONUM(A1)` = 10 (if cell A1 contains the label 'a)

@HLOOKUP

Format @HLOOKUP(*X*,*Block*,*Row*)

X = a numeric or string value
Block = a 2-D block value
Row = a numeric value ≥ 0

@HLOOKUP provides an efficient way to access information stored in a table. The first row of *Block* contains index values (comparison figures used to determine which column to search). @HLOOKUP searches horizontally through the first row of *Block* for value X. When @HLOOKUP finds X, it counts down that column until it reaches the specified *Row* and returns the value of that cell. The first row has a value of 0; the second row has a value of 1, and so on.

For example, you might have a table listing days of the week by number:

0	1	2	3	4	5	6
Saturday	Sunday	Monday	Tuesday	Wednesday	Thursday	Friday

You could use @HLOOKUP to enter the day of the week for a given date. For example,

`@HLOOKUP(@MOD(@DATE(93,7,11),7),A1..G2,1)` = Sunday

X can be either a character string or a number, the address or block name of any cell containing a label or value, or any expression that results in a number or string. If X is a string, Quattro Pro

looks for an exact match of either uppercase or lowercase. If *X* is a number and Quattro Pro can't find an equal number, it locates the highest number in the row not more than *X*.

The second argument, *Block*, specifies the coordinates of the table to be used for the lookup. This table must have its *index* values in the first row. Each cell of the index row must contain a value; @HLOOKUP returns 0 if the referenced cell is blank. If numbers, these values must be in ascending order.

Quattro Pro searches through the index row of the table from left to right looking for a match to *X*. If an exact match is found, the search stops at that column. If an exact match isn't found, the search stops at the value closest to but not greater than *X*. If *X* is a label, the lookup is successful only if an exact match is found. If *X* is a number and the index row contains only labels, the search stops at the rightmost column.

The final argument *Row* indicates how many rows down in the column the return value is located. This argument can be any value from 0 up to one less than the number of rows in the table, or any expression or cell address resulting in such a value. A value of 0 indicates to return the actual value found in the index row X. A value of 1 indicates to return the value directly below the value found in the index row; 2 indicates to return the value two rows below, and so on.

Note If *X* and the index row entries are string values and *Row* = 0, Quattro Pro returns the offset number of the column in which *X* is found, not the value of *X*.

Any of these instances result in ERR:

- *Row* is less than 0 or greater than the number of rows minus 1 in *Block*.
- *X* is less than the smallest value in the topmost row of *Block*.
- *X* and the index row entries are string values and Quattro Pro fails to find a match in the top row of *Block*.
- *X* is a string or label and the index row entries are numeric values.

Examples These examples refer to data in the next figure. In the first example, Quattro Pro searches across the first row of the specified block (row 1), looking for the largest number equal to or less than 17. It stops at cell D1, then moves down the specified number of rows (3). It stops at cell D4 and returns the value 47.

`@HLOOKUP(17,A1..H4,3)` = 47
`@HLOOKUP(10,A1..H4,0)` = 10
`@HLOOKUP(6,A1..H4,2)` = 42
`@HLOOKUP(50,A1..H4,3)` = 43
`@HLOOKUP("string",A1..H4,3)` = ERR ("string" is not an index row value)
`@HLOOKUP("Code",A1..H4,2)` = YXFG
`@HLOOKUP(18,A1..H4,4)` = ERR (row value > # rows-1 in block)
`@HLOOKUP(18,A1..H2,3)` = ERR (row value > # rows-1 in block)

Horizontal lookup table

	A	B	C	D	E	F	G	H
1	1	5	10	15	20	25	30	Code
2	43	53	32	67	45	45	90	XXFX
3	92	42	18	22	32	32	89	YXFG
4	45	83	76	47	35	35	43	LLGR

To search vertically through a table, use @VLOOKUP (page 103).

@HOUR

Format @HOUR(*DateTimeNumber*)

DateTimeNumber = a number between –109,571 (January 1, 1600) and 474,816.9999999 (December 31, 3199)

@HOUR returns the hour portion of *DateTimeNumber*. *DateTimeNumber* must be a valid date/time serial number. Because only the decimal portion of a serial number pertains to time, the integer portion of the number is disregarded. The result is between 0 (12:00AM) and 23 (11:00PM).

To extract the hour portion of a string that is in time format (instead of serial format), use @TIMEVALUE with @HOUR to translate the time into a serial number. To return standard hours (1–12) instead of military hours (1–24), use @MOD (page 72) with a parameter of 12.

(See also @TIME on page 99 and @TIMEVALUE on page 99.)

Examples
`@HOUR(.25)` = 6
`@HOUR(.5)` = 12
`@HOUR(.75)` = 18
`@HOUR(@TIMEVALUE("10:08am"))` = 10
`@MOD(@HOUR(@TIMEVALUE("9:31:52 PM")),12)` = 9

@IF

Format @IF(*Cond*,*TrueExpr*,*FalseExpr*)

Cond = a logical expression representing the condition to be tested

TrueExpr = a numeric or string value representing the value to use if *Cond* is true

FalseExpr = a numeric or string value representing the value to use if *Cond* if false

@IF evaluates the logical condition given as *Cond*. If the condition is found to be true, it returns the value given as *TrueExp*. If the condition is false, it returns the value given as *FalseExpr*. *Cond* is true if it evaluates to any nonzero numeric value.

The formula entered as *Cond* can be any logical expression that can be evaluated as true or false; for example, `B6<0` or `C3*D2=53`.

You can use compound conditions by connecting expressions with #AND# or #OR#. If you use #AND#, both conditions given must be met to evaluate true. If you use #OR#, the expression is true if either of the conditions is met. For example, `A3<10#OR#A3>5` means that the value in A3 must be *either* less than 10 *or* greater than 5 to evaluate true—6, 9, 1, and 15 are all true; `A3<10#AND#A3>5` means that the value in A3 must be *between* 6 and 9 to evaluate true.

You can also use the #NOT# operator to negate a condition. For example, `#NOT#(B3>10)` evaluates true if B3 is *not* greater than 10.

The expressions given as *TrueExpr* and *FalseExpr* can be either numeric values or text. Numeric values can be numbers or formulas resulting in numbers. If text is used, the string must be enclosed by double quotes; for example:

```
@IF(D6=5,"John","Harry")
```

You can also use cell references to use the contents of other cells in the notebook. For example, `@IF(B10<18,D5,C4)` enters the contents of D5 if the condition is true, and enters the contents of C4 if the condition is false.

If the condition you specify with *Cond* searches a cell for a number and the cell contains a label, the label is evaluated as having a value of 0 and *FalseExpr* is returned. Likewise, if you search for a label and find a numeric value, *TrueExpr* results if the value of the referenced cell is 0; *FalseExpr* results if it is nonzero.

Although logical expressions typically reference other cells, this is not required. Any expression resulting in a numeric value is accepted; for example, `A1=1` or `A1="Fred"`. If the result of *Cond* is nonzero, *TrueExpr* is the result; otherwise, *FalseExpr* is the result.

@IF statements can be *nested*, or used within one another. In other words, *TrueExpr* can contain yet another test to further validate *Cond*. For example, `@IF(B5>C6,@IF(B5>C7,1,2),3)` tells Quattro Pro to see if the contents of B5 are greater than C6. If they are, it then checks to see if B5 is greater than C7; if so, it enters a 1 in the cell. If not, it enters a 2. If B5 is *not* greater than C6, it enters a 3. There's no limit on the number of levels @IF expressions that you can nest, as long as the entire expression doesn't exceed 1024 characters.

Examples

`@IF(8=7,4,5)` = 5
`@IF(B4<100,"Yes","No")` = Yes if B4 < 100; otherwise, No
`@IF(C10=BLOCK,45,50)` = 45 if C10 = the cell named BLOCK; otherwise, 50
`@IF(C10,1,0)` = 0 if C10 = 0; otherwise, 1
`@IF(D:C10=6,"T","F")` = T if cell C10 on page D of the active notebook = 6; otherwise, F
`@IF([SALES]D:C10=6,"T","F")` = T if cell C10 on page D of notebook SALES = 6; otherwise, F
`@IF(A1>A2#AND#A1>A3,"True","False")` = True if A1=4, A2=3, and A3=2
`@IF(A1>A2#AND#A1>A3,"True","False")` = False if A1=4, A2=5, and A3=2
`@IF(A1>A2#OR#A1>A3,"True","False")` = True if A1=4, A2=5, and A3=2

@INDEX

Format @INDEX(*Block*,*Column*,*Row*,<*Page*>)

Block = a block value
Column = a numeric value ≥ 0
Row = a numeric value ≥ 0
Page = a numeric value ≥ 0

@INDEX, like the @LOOKUP functions, is used with blocks or tables of data. It searches through the table given as *Block* and returns the value specified with the *Column*, *Row*, and optional *Page* values. The top left cell in *Block* is column 0, row 0, page 0. The *Column*, *Row*, and *Page* values are not the actual coordinates of the

resulting cell, but instead are *offset* values. In other words, @INDEX begins in the top left cell of the given block, moves right the number of columns specified by *Column*, moves down the number of rows specified by *Row*, and through the number of pages specified by *Page*. It then returns the value in the current cell.

The *Column*, *Row*, and *Page* values must be numbers greater than or equal to zero and less than the number of rows, columns, or pages in the block. If a fractional number is used (for example, 2.35), the fractional part is dropped (*not* rounded).

Examples These examples reference cells in the data table.

`@INDEX(A1..G4,4,3)` = 22
`@INDEX(A1..G4,2,3)` = 44
`@INDEX(C2..E4,1,2)` = 33
`@INDEX(C2..E4,3,3)` = ERR (too many rows)
`@INDEX(A1..G4,-2,3)` = ERR (negative column number)

Data table

	A	B	C	D	E	F	G	H	I
1	1	2	3	4	5	6	7		
2	30	43	35	98	47	48	43		
3	14	15	19	84	45	34	23		
4	66	55	44	33	22	11	10		
5									

@INT

Format @INT(*X*)

X = a numeric value

@INT drops the fractional portion of *X*, returning its integer value. See @ROUND (page 89) for an @function that rounds *X* to the nearest integer.

Examples

`@INT(499.99)` = 499
`@INT(0.1245)` = 0
`@INT(-2.3)` = –2
`@INT(C4)` = 5 if C4 contains a value between 5 and 6

@IPAYMT

Format @IPAYMT(*Rate*,*Per*,*Nper*,*Pv*,<*Fv*>,<*Type*>)

Rate = a numeric value > –1, representing the fixed periodic interest rate
Per = identifies the payment period (where *Nper* is the number of periods)
Nper = a numeric value > 0, representing the number of periods of the loan
Pv = a numeric value representing the amount borrowed (the principal)
Fv = a numeric value representing the future value of the investment
Type = an optional argument indicating whether the cash flows occur at the beginning or end of the period

The last two arguments, *Fv* and *Type*, are optional. If you omit one or both of them, their values are assumed to be zero.

@IPAYMT and @PPAYMT tell what portion of a loan payment is interest and what portion is principal, respectively. Given *Rate*, *Nper*, *Pv*, *Fv*, and *Type*, the monthly payment (if we use a month as the example time period) is given by @PAYMT(*Rate*, *Nper*, *Pv*, *Fv*, *Type*)—see page 78 for more information on @PAYMT. For each month in the transaction period,

@PAYMT(*Rate*, *Nper*, *Pv*, *Fv*, *Type*)
= @IPAYMT(*Rate*, *Per*, *Nper*, *Pv*, *Fv*, *Type*)
+ @PPAYMT(*Rate*, *Per*, *Nper*, *Pv*, *Fv*, *Type*)

where *Per* identifies the *Per*th month, *Per* = 1, ..., *Nper*. @IPAYMT is calculated by computing the simple interest on the outstanding principal from the previous month. @PPAYMT then gives the principal portion of the payment for the current month, and is computed by subtracting @IPAYMT from @PAYMT. The calculation starts by using *Pv* as the outstanding principal at the beginning of the first month.

@IPAYMT(*Rate*, *Per*, *Nper*, *Pv*, *Fv*, *Type*)
= *Rate* * @FVAL(*Rate*, *Per* + (*Type* – 1),
@PAYMT(*Rate*, *Nper*, *Pv*, *Fv*, *Type*), *Pv*, *Type*)

Examples If you are two years into a 30-year, 10% mortgage on a $100,000 loan and your interest payment is tax-deductible, then

Remember, a negative result simply means the money is out of your pocket and not coming to you.

```
@IPAYMT(.1/12,2*12,30*12,100000)
```

returns your current month's deduction, or –824.03.

@IRATE

Format @IRATE(*Nper*,*Pmt*,*Pv*,<*Fv*>,<*Type*>)

Nper = a numeric value > 0 representing the length of the investment in terms of the number of compounding periods
Pmt = a numeric value representing the amount of the periodic payment
Pv = a numeric value representing the current value of an investment
Fv = a numeric value representing the future value of an investment
Type = a numeric value that indicates whether the cash flows occur at the beginning or end of the period

Be sure to enter a negative number for money that's out of your pocket and a positive number for money that's coming in to you.

@IRATE is another version of the 1-2-3–compatible @function @RATE (page 86). @IRATE calculates the interest rate required to pay off a principal (*Pv*) or reach an investment goal (*Fv*) in *Nper* payments of *Pmt* amount.

The last two arguments, *Fv* and *Type*, are optional. If you omit one or both of them, their values are assumed to be zero. See "Financial functions" on page 13 for more information.

@IRATE requires that the initial cash flow (*Pv* + *Type* * *Pmt*) and the last cash flow (*Fv* + (1-*Type*) * *Pmt*) have opposite signs. Otherwise, @IRATE returns ERR because the transaction is not simple and there may not be a meaningful rate.

Examples Assume you're negotiating to buy a $15,000 new car. The sales-person says you can have the car for $500 a month for the next five years. To calculate the monthly percentage rate:

`@IRATE(5*12,-500,15000,0,0)` = 0.02632

Another example: Assume that you plan to deposit $2000 a year into a savings account that currently contains only $2.38. What

interest rate must the account earn to generate $15,000 at the end of 5 years? Here's how to calculate this:

`@IRATE(5,-2000,-2.38,15000,0)` = 0.2038

@IRR

Format @IRR(*Guess*,*Block*)

Guess = a numeric value that estimates the internal rate of return on an investment
Block = a block that contains cash flow amounts for the investment

@IRR determines the internal rate of return on an investment. It references a block in your notebook that contains cash flow information and uses the supplied internal rate of return estimate to calculate the results.

Before you use this @function, you must set up a cash flow table, showing expected cash flow amounts over a period of time. Quattro Pro assumes that the amounts are received at regular intervals. Negative amounts are interpreted as cash outflows, and positive amounts as inflows. The first amount must be a negative number, to reflect the initial investment. These amounts can all be the same for each time period, or they can be different (including a mixture of negatives, positives, or zeros).

The next section discusses simple transactions, which require no *Guess* value. "Multiple rates of return," on page 63, discusses how to use @IRR to find the rate of return for a transaction with two or more roots.

Simple transactions

@IRR always returns ERR or a rate of return greater than or equal to –1. Some cash flows have no rate of return, and some have several. If it can be determined that the cash flow has a unique rate of return, then the *Guess* is ignored and @IRR gives that unique value.

Quattro Pro can determine unique rates of return for *simple transactions* or cash flow. A simple cash flow has two sets of values: a series of nonpositive values (negative values and zero) that is your cash outflow, and a series of nonnegative values (positive values and zero) that is your cash inflow. A simple cash

flow must contain both a negative value (cash outflow) and a positive value (cash inflow).

Simple values have a unique rate of return, so you can safely use @NA as the *Guess* argument in most cases. Quattro Pro then tries to determine whether the root is unique and returns that rate without using a *Guess* argument. If Quattro Pro can't find a unique root value, @IRR returns ERR. (See "Multiple rates of return" on page 63 for cases with multiple roots.)

Typically, you make an investment (a negative cash flow) and then receive several dividends (positive cash flows). This is an example of a simple transaction, and @IRR gives the unique rate of return for this without requiring a *Guess*. More complex transactions, in which the direction of money changes several times, often do not have a meaningful value for @IRR.

@IRR(*Guess*,*Block*) gives the number *Rate* which satisfies @NPV(*Rate*,*Block*,0) = 0. For a simple transaction, @NPV(@IRR(*Block*),*Block*,0) will give a number close to 0 (it may not be exactly 0 due to the way numbers are rounded off).

Guess can be any value greater than –1. Values that are NA or less than or equal to –1 are ignored. Use @NA for *Guess* unless you read and understand the next section about multiple rates of return. If you use @NA, you will get ERR if your cash flow has more than one rate of return, rather than the rate of return that happens to be near your *Guess*.

There is no *Type* parameter to @IRR because the rate of return is the same regardless of whether the payments are at the end or the beginning of each period.

Multiple rates of return

In unusual cases, @IRR may have as many as *N-1* roots, where *N* is the number of terms in the block. Consider the *Block* that has the values (–10, +150, –145). @IRR(@NA,*Block*) returns ERR because it is not simple. The two roots are 3.86% and 1296%, obtainable from guesses of 0 and 10, respectively. Both of these values are meaningful, if interpreted properly.

Maybe you invested $10 in an oil well. It struck oil, paying you $150, but then it ran into legal difficulties and required you to pay back $145. You had a net loss of $5, but your rate of return is quite large, as you had the use of a relatively large amount of money for a small investment. Or, maybe the real purpose of the transaction was to get a $150 loan from the bank. The bank required you

to pay a $10 application fee ahead of time. After you got the loan, you paid back $145. Because you only paid back $155 on a $150 loan, the interest rate is fairly low. The difference in these two interpretations is that in one you're the lender, and in the other you're the borrower.

If you find a transaction with two roots, there is a mechanical way to determine which is the lender rate and which is the borrower rate. Pick a positive term in the *Block*, and increase it by a small amount. If the rate increases, it is a lender rate, and if the rate decreases, it is a borrower rate.

If *Block* has the values (-10,+150,-145), @IRR(0,*Block*) = 3.86%
If *Block* has the values (-10,+150,-145), @IRR(10,*Block*) = 1296%
If *Block* has the values (-10,+150.1,-145), @IRR(0,*Block*) = 3.78%
If *Block* has the values (-10,+150.1,-145), @IRR(10,*Block*) = 1297%

Because 3.78% < 3.86% and 1297% > 1296%, it follows from this rule that 3.86% is a borrower rate and 1296% is a lender rate.

Most uses of @IRR are for analyzing an investment in which the first cash flow is negative, and the rate is a lender rate.

Some transactions have no rate of return at all. @IRR(*Guess*,*Block*), with *Block* having the values (-1,+1,-1), returns ERR regardless of the *Guess*. There is no rate of return that is meaningful for this cash flow.

If there are more than two roots, the above analysis can still be used to determine if a particular root is a lender rate or a borrower rate. In some cases, it might still be possible to assign meaning to a root, but it is much more likely that the transaction should be interpreted as several transactions, with a rate of return for each. For example, the cash flow (-1,+6,-11,+6) has three roots, 0%, 100%, 200%. It is difficult to interpret such a transaction in terms of interest rates, and the roots are sensitive to small fluctuations.

Examples These examples reference the cash flow table in the next figure.

`@IRR(0,A1..A7)` = 16.85%
`@IRR(0,B1..B7)` = –20.60%
`@IRR(0,C1..C7)` = 4.70%

Cash flow table

	A	B	C	D	E	F	G
1	-3000	-50000	-10000				
2	700	-8000	1000				
3	600	2000	1000				
4	750	4000	1200				
5	900	6000	2000				
6	1000	5000	3000				
7	1400	4500	4000				

@ISERR

Format @ISERR(*X*)

X = a cell address or expression

@ISERR is normally used to check the contents of a cell for errors. If the cell contains ERR, 1 is returned; otherwise, 0 is returned. You can also use formulas or numeric values with @ISERR.

When an error occurs in a cell, any formula referencing that cell results in ERR, and any cell referencing that cell becomes ERR. This sometimes causes the value of ERR to ripple through your notebook. @ISERR, used with @IF, can halt this ripple effect, as demonstrated next.

In the notebook, sales for each division are divided by the number of sales reps to determine average sales per rep for each division. The West Division is late with its figures, which would normally cause an ERR value to be displayed both for that division's average and for the overall average (you cannot divide by zero). In this case, however, @ISERR is used with @IF to enter " – –" for the division's average if the division's sales figures are blank.

D3: @IF(@ISERR(B3/C3),"- -",B3/C3)

Trapping errors with @IF and @ISERR

	A	B	C	D	E	F
1	DIVISION	SALES	REPS	AVG		
2	North	$252,933	37	$6,836		
3	West	—		–		
4	South	$198,355	24	$8,265		
5	East	$309,422	35	$8,841		
6						
7			Overall Avg:	$5,985		
8						

Examples
`@ISERR(C2)` = 1 if C2 contains ERR; otherwise, 0
`@ISERR(10/0)` = 1
`@ISERR(45+C3)` = 1 if C3 is ERR; otherwise, 0
`@ISERR(C2/B3)` = 1 if B3 is 0 or ERR, or if C2 is ERR; otherwise, 0
`@IF(@ISERR(A2),0,A5)` = 0 if A2 is ERR; otherwise, it returns the value in A5

@ISNA

Format @ISNA(X)

X = a cell address or expression

@ISNA tests for the special value NA in a cell. If the cell contains an NA value, it returns 1; otherwise, it returns 0. NA is considered a special value; it appears in the notebook only through the use of @NA (page 74). Cells containing the label "NA" (not produced by @NA) are not recognized by @ISNA.

Examples
`@ISNA("NA")` = 0
`@ISNA(@NA)` = 1
`@ISNA(A18)` = 1 if A18 contains NA produced by @NA

@ISNUMBER

Format @ISNUMBER(X)

X = a cell address or expression

@ISNUMBER examines X and determines if it contains a numeric value. If X is blank or contains a numeric value, ERR, or NA, it returns a 1. If X is a label or text, it returns a 0. @ISNUMBER is usually used with @IF to determine whether an entry is a value.

Examples
`@ISNUMBER(88)` = 1
`@ISNUMBER("88")` = 0 (quotes signify a text string)
`@ISNUMBER(9/15/95)` = 1
`@ISNUMBER(@ERR)` = 1 (ERR and NA are numeric values)

@ISSTRING

Format @ISSTRING(*X*)

X = a cell address or expression

@ISSTRING examines *X* and determines if it contains a label or text string. If it does (even if it's an empty text string), it returns a 1. If it is blank or contains a numeric or date value, it returns a 0.

Usually, @ISSTRING is used to test the contents of a cell. You can test any expression, however. Literal string arguments must be enclosed by double quotes.

Examples

`@ISSTRING(55)` = 0
`@ISSTRING(2/5/88)` = 0
`@ISSTRING("Hello, world.")` = 1
`@ISSTRING("Hello, "&"world.")` = 1
`@ISSTRING("55")` = 1
`@ISSTRING(A15)` = 1 if A15 contains a label, otherwise 0
`@ISSTRING(A15&A16&"!!!")` = 1 if A15 and A16 contain labels of formulas that result in strings
`@ISSTRING("")` = 1 ("" is an empty string)
`@ISSTRING(@NA)` = 0 (NA and ERR are values)

@LEFT

Format @LEFT(*String*,*Num*)

String = a string value
Num = a numeric value ≥ 0

@LEFT returns the leftmost *Num* characters of *String*. It lets you extract a specified number of characters from the left side of a string or label.

If *String* is a numeric or date value or a blank cell, @LEFT returns ERR. If *Num* is longer than the length of *String*, all of *String* is returned. The number of characters returned is never greater than the length of the string.

@MID and @RIGHT are related @functions.

Examples

`@LEFT("Jennifer",5)` = Jenni
`@LEFT("Jennifer",15)` = Jennifer
`@LEFT("155",1)` = 1

`@LEFT("     Jennifer",6) =` J (including five leading spaces)
`@LEFT(123,1)` = ERR (123 is a value)
`@LENGTH(@LEFT("Jennifer",255))` = 8

@LENGTH

Format @LENGTH(*String*)

String = a string value

@LENGTH returns the number of characters in *String*, including spaces. You can combine strings or cell addresses with an ampersand (&). When *String* is a text string, it must be enclosed by double quotes.

If you try to reference a blank cell with this @function, Quattro Pro returns ERR.

Examples `@LENGTH("Hello, world.")` = 13
`@LENGTH(" Jennifer")` = 9 (including preceding space)
`@LENGTH("Greetings "&"earthling")` = 19 (including space after Greetings)
`@LENGTH(29584949)` = ERR (29584949 is a value, not a string)
`@LENGTH(A6&B10)` = total number of characters in A6 and B10
`@LENGTH(B10)` = ERR (if B10 is blank)

@LN

Format @LN(*X*)

X = a numeric value > 0

@LN returns the natural logarithm of *X*. A natural logarithm uses the mathematical constant *e* as a base. @LN produces the inverse of @EXP (page 50).

Examples `@LN(3)` = 1.098612289
`@LN(@EXP(10))` = 10
`@LN(16)/@LN(2)` = 4
`@LN(-4)` = ERR (-4 is less than 0)

@LOG

Format @LOG(*X*)

X = a numeric value > 0

@LOG returns the base 10 logarithm of *X*.

Examples `@LOG(1000)` = 3
`@LOG(10^23.8)` = 23.8
`@LOG(16)/@LOG(2)` = 4

@LOWER

Format @LOWER(*String*)

String = a string value

@LOWER returns *String* in lowercase characters. Numbers and symbols within a string are unaffected. Numeric and date values return ERR.

Examples `@LOWER("UPPER")` = upper
`@LOWER("Hello, world.")` = hello, world.
`@LOWER("145 Bancroft Lane")` = 145 bancroft lane
`@LOWER(4839)` = ERR
`@LOWER(@LEFT("Johnson",1))` = j

@MAX

Format @MAX(*List*)

List = one or more numeric or block values

@MAX returns the largest numeric or date value in *List*. If more than one block is listed, commas must separate the blocks. If any of the cells referenced contain ERR, the resulting value is ERR.

Examples The examples refer to cells in the next figure.

`@MAX(B3..B8)` = $750
`@MAX(C3..C8,E3..E8)` = $833
`@MAX(A1..E10)` = $2,500
`@MAX(B3..C8,E3)` = $833

Monthly sales report

	A	B	C	D	E	F	G
1							
2		JANUARY	FEBRUARY	MARCH	APRIL		
3	JA	$652	$833	$599	$734		
4	MH	$456	$305	$522	$478		
5	RB	$68	$59	$73	$79		
6	PD	$379	$379	$379	$379		
7	AH	$80	$80	$80	$80		
8	MS	$750	$750	$750	$750		
9							
10	TOTAL	$2,385	$2,406	$2,403	$2,500		
11							

@MEMAVAIL

Format @MEMAVAIL

@MEMAVAIL returns the number of bytes of memory currently available.

Examples `@MEMAVAIL` = 47819 (if 47,819 bytes of memory are available)

@MEMEMSAVAIL

Format @MEMEMSAVAIL

This @function is included for compatibility with Quattro Pro for DOS; it always returns NA under Windows.

@MID

Format @MID(*String*,*StartNumber*,*Num*)

String = a string value
StartNumber = a numeric value ≥ 0
Num = a numeric value ≥ 0

@MID extracts the first *Num* characters of *String* starting at character number *StartNumber*, which is the number of characters to the right of the first character (character 0). It is similar to the @LEFT, which extracts *Num* characters of *String* beginning with the first character. The difference is that you can specify a character other than the first character in the string.

String can be any text string (enclosed by quotes) or reference to a cell containing a label. If *StartNumber* is greater than or equal to the length of *String* or if *Num* is 0, the result is "", or an empty string.

Examples `@MID("Abraham Lincoln",8,7)` = Lincoln
`@MID("George Washington",7,4)` = Wash
`@MID("Theodore Roosevelt",19,5)` = ""
`@MID(A23,@FIND("Roosevelt",A23,0),@LENGTH( "Roosevelt"))` =
Roosevelt (if A23 = Franklin Roosevelt)

@MIN

Format @MIN(*List*)

List = one or more numeric or block values

@MIN returns the smallest numeric value in *List*. If *List* contains more than one value, commas must separate the values. Labels are treated in all statistical functions as 0 and should therefore be excluded from *List*.

If *List* is entered as a block and one or more cells in that block are blank, the blanks are excluded from the calculation; otherwise, blanks are treated as 0.

Examples The examples refer to cells in the next figure.

`@MIN(B3..B8)` = $68
`@MIN(B5..E5,B7..E7)` = $59
`@MIN(B3..E3)` = $599

Monthly sales report

	A	B	C	D	E	F	G
1							
2		JANUARY	FEBRUARY	MARCH	APRIL		
3	JA	$652	$833	$599	$734		
4	MH	$456	$305	$522	$478		
5	RB	$68	$59	$73	$79		
6	PD	$379	$379	$379	$379		
7	AH	$80	$80	$80	$80		
8	MS	$750	$750	$750	$750		
9							
10	TOTAL	$2,385	$2,406	$2,403	$2,500		
11							

@MINUTE

Format @MINUTE(*DateTimeNumber*)

DateTimeNumber = a number between –109,571 (January 1, 1600) and 474,816.9999999 (December 31, 3199)

@MINUTE returns the minute portion of *DateTimeNumber*. *DateTimeNumber* must be a valid date/time serial number. Because only the decimal portion of a serial number pertains to time, the integer portion of the number is disregarded. The result is between 0 and 59.

To extract the minute portion of a string that is in time format (instead of serial format), use @TIMEVALUE with @MINUTE to translate the time into a serial number (see page 99). You can also use @TIME to enter a time value instead of a serial number (see page 99).

Examples

`@MINUTE(.36554)` = 46
`@MINUTE(.2525)` = 3
`@MINUTE(35)` = 0
`@MINUTE(@TIME(3,15,22))` = 15
`@MINUTE(@TIMEVALUE("10:08 am"))` = 8

@MOD

Format @MOD(*X*,*Y*)

X = a numeric value
Y = a numeric value ≠ 0

@MOD divides the *X* value by *Y* and returns the remainder value, or modulus. Because you cannot divide a number by zero, ERR results if the value of *Y* is zero.

Examples

`@MOD(3,1)` = 0 (3 divided by 1 leaves no remainder)
`@MOD(5,2)` = 1 (5 divided by 2 leaves a remainder of 1)
`@MOD(3,1.1)` = 0.8
`@MOD(4,0)` = ERR

@MONTH

Format @MONTH(*DateTimeNumber*)

DateTimeNumber = a number between –109,571 (January 1, 1600) and 474,816.9999999 (December 31, 3199)

@MONTH returns the month portion of *DateTimeNumber*. *DateTimeNumber* must be a valid date/time serial number. Only

the integer portion is used. The result is between 1 (January) and 12 (December).

To extract the month portion of a string that is in date format (instead of serial format), use @DATEVALUE with @MONTH to translate the date into a serial number (see page 35). You can also use @DATE to enter a date value instead of a serial number (see page 34).

To display the *name* of the resulting month instead of the number, use a lookup table with the numbers corresponding to their names as shown in the example:

Month lookup table

	A	B	C	D	E	F	G
1	1	January					
2	2	February					
3	3	March					
4	4	April					
5	5	May					
6	6	June					
7	7	July					
8	8	August					
9	9	September					
10	10	October					
11	11	November					
12	12	December					
13							

You could then use @VLOOKUP (see page 103) to display the corresponding month name:

@VLOOKUP(@MONTH(*DateTimeNumber*),$A$1..$B$12,1)

Examples

```
@MONTH(69858) = 4
@MONTH(58494) = 2
@MONTH(.3773) = 12
@MONTH(@DATEVALUE("3/5/88")) = 3
@MONTH(@DATE(88,3,5)) = 3
@MOD(@MONTH(@DATEVALUE("3/5/88")),12) = 3
```

@N

Format @N(*Block*)

Block = a block value

@N inspects *Block* and returns the numeric value of the upper left cell. If that cell contains a label or is blank, it returns a 0.

This @function is used by other spreadsheet programs to avoid unnecessary ERR values resulting from labels included in calculations. This is unnecessary with Quattro Pro, however, because labels are already considered zero values in calculations. @N is included in Quattro Pro only for compatibility with other products.

@NA

Format @NA

@NA returns the special value NA (Not Available). Formulas that depend on a value entered as @NA return the value NA, unless there is an error, in which case they return ERR. NA is a unique number, not to be confused with the label NA.

@NA is used to indicate values not yet available (it won't work with labels). It ensures that formulas relying on information that is not provided don't display inaccurate data.

In the example, @NA has been entered for the South's Qtr 4 results. As you can see, the NA cascades through to the totals. When the @NA is replaced with a valid value, the totals will immediately reflect the correct figures.

@NA used as a placeholder

	A	B	C	D	E
1					
2	DIVISION	NORTH	SOUTH	WEST	
3	Qtr 1	$187,681	$151,136	$131,123	
4	Qtr 2	$170,072	$197,751	$149,181	
5	Qtr 3	$151,374	$102,791	$111,311	
6	Qtr 4	$118,213	NA	$194,456	
7					
8					
9	YTD	$627,340	NA	$586,071	
10	Average	$156,835	NA	$146,518	
11					

Examples `@NA` = NA
`@IF(B6=0,@NA,B6)` = NA if B6 = 0; otherwise, the value of B6

@NOW

Format @NOW

@NOW returns the serial number corresponding to the current date and time. To display the number as a date, choose Numeric Format in the block Object Inspector.

The value generated by @NOW is updated to the current date and time each time you recalculate the notebook.

The integer part of a date/time serial number represents the date; the decimal portion represents time. To extract just the date portion, use @INT(@NOW) or @TODAY. To extract just the time portion, use @MOD(@NOW,1).

Examples

`@NOW` = 33625.5000 (1/22/92, 12:00 PM)
`@INT(@NOW)` = 33625 (1/22/92)
`@MOD(@NOW,1)` = 0.5000 (12:00 PM)
`@INT(@MOD(@NOW,7))` = 4 (the number of the day of the week)

@NPER

Format @NPER(*Rate,Pmt,Pv,<Fv>,<Type>*)

Rate = a numeric value > –1 representing the fixed interest rate per compounding period
Pmt = a numeric value representing the amount of the periodic payment
Pv = a numeric value representing the present value of the investment
Fv = a numeric value representing the value that the investment will reach at some point
Type = a numeric value (0 or 1) that indicates whether the cash flows occur at the beginning (1) or the end (0) of the period

Be sure to enter a negative number for money that's out of your pocket and a positive number for money that's coming in to you.

@NPER is another version of the 1-2-3–compatible @functions @CTERM and @TERM, which compute the number of payment periods required for an investment of *Fv* using two different sets of arguments (see pages 32 and 98, respectively). The last two arguments of @NPER, *Fv* and *Type*, are optional. If you omit the last one or both of them, Quattro Pro assumes their values are zero. See "Financial functions" on page 13 for more information about the relationship between these @functions.

Examples Assume you have an IRA account that earns 11.5% interest paid annually at the start of the year, and you deposit $2000 into the account at the end of each year. The present account balance is $633. To determine how many payment periods it will take to reach a nest egg of $50,000, use @NPER:

`@NPER(11.5%,-2000,-633,50000,0)` = 12.12

(The fractional part of the answer is not very meaningful; you can't be sure of having your nest egg until the end of the 13th year.)

@NPV

Format @NPV(*Rate*,*Block*,<*Type*>)

Rate = a numeric value representing a fixed periodic interest rate
Block = a block containing expected cash flow information
Type = an optional argument indicating whether the cash flows occur at the beginning or end of the period

@NPV calculates the current value of a set of estimated cash flow values (*Block*), discounted at the given interest rate (*Rate*). It is helpful in determining how much an investment is currently worth, based on expected earnings, although its accuracy is entirely dependent on the accuracy of the cash flow table.

@NPV has an optional third argument, *Type*, which is not compatible with 1-2-3. *Type* can be 0 or 1, depending on whether the cash flows are at the beginning or the end of the period. (This use of *Type* is the same as for the other financial functions. As with the other financial functions, the default value is 0. See "Financial functions" on page 13 for more information.)

The formula for @NPV(*Rate*,*Block*,*Type*)—if *Block* consists of $v_1, \ldots, v_n$—is given by

If *Type* = 0:

$$\frac{v_1}{(1 + Rate)} + \ldots + \frac{v_n}{(1 + Rate)^n}$$

If *Type* = 1:

$$V_1 + \frac{V_2}{(1 + Rate)} + \ldots + \frac{V_n}{(1 + Rate)^{n-1}}$$

The cash flow table you reference should show expected income and debits over a period of time. If *Type* is 0, Quattro Pro assumes that the amounts are received at the end of regular intervals. If *Type* is 1, it assumes the amounts are received at the beginning of regular intervals. Quattro Pro also assumes that the length of this interval is the same as the period on which interest is compounded. In other words, if monthly cash flow is estimated, *Rate* needs to show monthly interest. To convert annual interest to monthly interest, simply divide by 12.

Examples Suppose you're considering investing $5000 this year, and you expect a return of $2000 in each of the next four years. Put the values –5000,+2000,+2000,+2000,+2000 in the block A1..A5. The net present value, using a discount rate of 10%, is @NPV(.1,A1..A5,1) which equals $1,340. Alternatively, you can combine the initial investment with the present value of the four returns by using +A1+@NPV(.1,A2..A5,0). In either case, the result is the same.

These examples reference the cash flow table in the next figure.

`@NPV(1.25%,B1..B7)` = $62,683.77
`@NPV(15%/12,C1..C7)` = $3,507
`@NPV(15%/12,D1..D7)` = $31,216
`-2000+@NPV(15%/12,D1..D7)` = $29,215.77 (assumes an initial cash outflow of $2,000)

Cash flow table

	A	B	C	D	E	F
1	January	8000	200	3500		
2	February	9000	350	4000		
3	March	8500	-300	3000		
4	April	9500	600	5000		
5	May	10000	700	4000		
6	June	11000	1000	6500		
7	July	10000	1200	7000		
8						
9						

@NUMTOHEX

Format @NUMTOHEX(*X*)

X = a numeric value

@NUMTOHEX converts the decimal number *X* to its corresponding hexadecimal string value. @HEXTONUM, described on page 54, performs the opposite conversion, from hex to decimal.

Examples

```
@NUMTOHEX(10) = 'A
@NUMTOHEX(16) = '10
@NUMTOHEX(65535) = 'FFFF
```

@PAYMT

Format @PAYMT(*Rate*,*Nper*,*Pv*,<*Fv*>,<*Type*>)

Rate = a numeric value > –1, representing interest rate

Nper = a numeric value > 0, representing the number of periods of the loan

Pv = a numeric value representing the amount borrowed (the principal)

Fv = a numeric value representing the value that the investment will reach at some point

Type = a numeric value that indicates whether the cash flows occur at the beginning or the end of the period

Be sure to enter a negative number for money that's out-of-pocket and a positive number for money that's coming in.

@PAYMT is another version of the 1-2-3–compatible @function @PMT (page 79). The last two arguments of @PAYMT, *Fv* and *Type*, are optional. If you omit the last one or both of them, Quattro Pro assumes that their values are zero. See "Financial functions" on page 13 for more information about the relationship between @PAYMT and @PMT.

The @functions @IPAYMT (page 60) and @PPAYMT (page 81) give the parts of the payment that are interest and principal, respectively.

This @function has more versatility than @PMT in that it can calculate annuity due and can take into consideration the future value of the principal.

Examples Assume you want to take out a 30-year $175,000 mortgage with a 17.5% annual interest rate with 12 payments a year, and you'd like to see the difference in your monthly payments if you paid at the start or at the end of the month. All you have to do is enter these two @functions:

`@PAYMT(17.5%/12,12*30,175000,0,0)` = –2566.07
`@PAYMT(17.5%/12,12*30,175000,0,1)` = –2529.19

If, on the other hand, your mortgage has a so-called balloon payment that leaves you with a hefty amount of unpaid principal at the end of the mortgage, you can still calculate the payment. Just insert the balloon payment amount (say, $80,000) as the *future value* component:

`@PAYMT(17.5%/12,12*30,175000,-80000,0)` = –2559.68

@PI

Format @PI

@PI returns the value of π (3.141592653589794...), the classic ratio of a circle's circumference to its diameter.

To figure the area of a circle, given the radius in cell A1, enter this formula:

```
@PI*A1^2
```

Examples `@PI*13` = 40.84 (circumference of circle with a diameter of 13)
`@PI*(7.5)^2` = 176.7142 (area of circle with a radius of 7.5)
`@PI*B3` = the circumference of a circle whose diameter is in B3

@PMT

Format @PMT(*Pv,Rate,Nper*)

Pv = a numeric value representing the amount borrowed (the principal)

Rate = a numeric value > –1, representing the interest rate

Nper = a numeric value > 0, representing the number of periods of the loan

@PMT calculates the fully amortized payment of borrowing *Pv* dollars at *Rate* percent per period over *Nper* periods. It assumes that interest is paid at the end of each period.

Quattro Pro has a more useful and general @function to perform this calculation, @PAYMT, which is not compatible with 1-2-3. See the entry for @PAYMT on page 78 for details, as well as "Financial functions" on page 13.

@PMT uses this formula:

$$\frac{Pv * Rate}{1 - (1 + Rate)^{-Nper}}$$

An equivalent for this formula using @PAYMT is

@PAYMT(*Rate, Nper, – Pv,* 0)

You can enter the value for *Rate* as a decimal or a fraction; for example, 9.5% or .095. The amount you specify for *Rate* must correlate with the unit used for *Nper*. In other words, if payments are made and interest calculated annually, the amount entered for *Nper* must represent years. If monthly, *Nper* must represent the number of months the loan covers. To calculate monthly payments using an annual interest rate, divide the interest rate by 12.

@PMT assumes that the investment is an ordinary annuity. @PAYMT, @IPAYMT, and @PPAYMT, which are calculated differently than but are related to @PMT, let you use an optional argument (*Type*) to indicate whether the investment is an ordinary annuity or an annuity due. "Financial functions" on page 13 describes the relationship between these @functions.

Examples To calculate a monthly payment (paid at the last day of the month) for a three-year loan of $10,000 at an annual 15% interest rate, enter

`@PMT(10000,15%/12,3*12)` = $346.65

You can also use @PAYMT to figure this payment:

`@PAYMT(15%/12,3*12,10000,0,0)` = $-346.65

(The negative result means the payment is out of your pocket.)

Other examples:

`@PMT(1000,0.12,5)` = $277.41
`@PMT(500,0.16,12)` = $96.21
`@PMT(5000,16%/12,12)` = $453.65

`@PMT(12000,0.11,15)` = $1,668.78

`@PMT(10000,15%/12,36)` calculates a monthly payment for a three-year loan of $10,000 at an annual 15% interest rate

@PPAYMT

Format @PPAYMT(*Rate,Per,Nper,Pv,<Fv>,<Type>*)

Rate = a numeric value > –1, representing interest rate
Per = a numeric value, representing the number of periods into loan for which the principal is desired
Nper = a numeric value > 0, representing the number of periods of the loan
Pv = a numeric value representing amount borrowed (the principal)
Fv = a numeric value representing the value the investment will reach at some point
Type = a numeric value that indicates whether the cash flows occur at the beginning or end of the period

The @functions @IPAYMT (page 60) and @PPAYMT give the parts of the payment which are interest and principal, respectively.

See the entry for @IPAYMT for an explanation of how @IPAYMT and @PPAYMT interrelate, and how they both relate to @PAYMT.

Examples Assume you're two years into a 30-year, 10% mortgage on a $100,000 loan. To determine what portion of this month's payment is principal, enter

`@PPAYMT(.1/12,2*12,30*12,100000)` = $-53.54 (the result is negative to indicate the money is out of your pocket)

Another example:

`@PPAYMT (.15/4,24,40,10000,0,1)` = $-233.24 (quarterly payments for $10,000 loan at 15% annual percentage rate adjusted to a quarterly basis over a 10-year term)

@PROPER

Format @PROPER(*String*)

String = a string value

@PROPER converts the first letter of every word in *String* to uppercase, and the rest of the characters to lowercase. A word is defined as an unbroken string of alphabetic characters. Any blank spaces, punctuation symbols, or numbers mark the ending of a word.

Examples `@PROPER("GEORGE washINGTON")` = George Washington
`@PROPER("FIRST QUARTER")` = First Quarter
`@PROPER("JOHN J.  SMITH")` = John J. Smith
`@PROPER("1979's results")` = 1979'S Results
`@PROPER (A1)` = John J. Smith (where cell A1 contains JOHN J. SMITH)

@PROPERTY

Format @PROPERTY(*Object.Property*)

Object = the object whose property setting you want to retrieve (for example, Application or Block)
Property = the property whose setting you want to retrieve

@PROPERTY is similar to @COMMAND and @CURVALUE. It returns the current setting of *Property* for the requested *Object*.

Object.Property must be enclosed in double quotes. See the description of {SETOBJECTPROPERTY} on page 263 for details on *Object* and *Property* syntax. Appendix B lists each object and property you can enter as arguments.

@PROPERTY returns strings; even if the setting is a number, it is returned as a string.

@PROPERTY can be used in macros to read current settings and restore them at the end of the macro, after making changes during the macro.

Caution! @PROPERTY statements do not recalculate automatically as many other @functions do. Press *F9* to obtain the current value.

Examples `@PROPERTY("Active_Page.Name")` = Sales when the active page is named Sales.

`@PROPERTY("Active_Block.Selection")` = the coordinates of the currently selected block.

`@PROPERTY("Active_Block.Protection")` = Protect if the currently selected block is protected; otherwise, it returns Unprotect.

`@PROPERTY("Sales:A1..D12.Protection")` = Protect if block A1..D12 on page Sales is protected; otherwise, it returns Unprotect.

`@PROPERTY("Graph8:G$Pane.Fill_Style")` = Bitmap,Crop to fit,C:\QPW\tiger.bmp if the Graph Pane Fill Style for Graph8 is a cropped bitmap named Tiger from directory C:\QPW.

`@PROPERTY("B4.Font.Typeface")` = Courier when cell B4 of the active page is set to display Courier type.

@PV

Format @PV(*Pmt*,*Rate*,*Nper*)

Pmt = a numeric value representing the amount of the periodic payment
Rate = a numeric value > –1, representing periodic interest rate
Nper = a numeric value > 0 representing the number of payments to be made

@PV calculates the present value of an investment where *Pmt* is received for *Nper* periods and is discounted at the rate of *Rate* per period. Present value is calculated using this formula:

$$Pmt \frac{1 - (1 + Rate)^{-Nper}}{Rate}$$

An equivalent for this formula using @PVAL is

@PVAL(*Rate*, *Nper*, –*Pmt*, 0)

@PV assumes that the investment is an ordinary annuity. @PVAL, which is calculated differently than but is related to @PV, lets you use an optional argument (*Type*) to indicate whether the investment is an ordinary annuity or an annuity due. "Financial functions" on page 13 describes the relationship between @PV and @PVAL.

Examples For example, assume your neighborhood day-care center wants to buy a new van that costs $12,000. The car dealership presents two

offers: Pay the whole amount up front, or pay $350 per month for the next five years with 7% interest. The present value of that loan is

`@PV(350,7%/12,5*12)` = $17,675.70

This tells you that the loan is worth over $5000 more than paying the cost all at once.

You can also use @PVAL (described next). The car loan example could be rewritten as

`@PVAL(7%/12,5*12,-350,0,0)` = $17,675.70

Other examples:

`@PV(277,0.12,5)` = $998.52
`@PV(600,0.17,10)` = $2,795.16
`@PV(100,0.11,12)` = $649.24

@PVAL

Format @PVAL(*Rate*,*Nper*,*Pmt*,<*Fv*>,<*Type*>)

Rate = a numeric value > –1, representing periodic interest rate
Nper = number of periods, which should be an integer > 0
Pmt = a numeric value representing the amount of the periodic payment
Fv = future value
Type = 0 if payments are at the end of each period, 1 if they are at the beginning

Be sure to enter a negative number for money that's out of your pocket and a positive number for money that's coming in to you.

@PVAL is another version of the 1-2-3-compatible @function @PV (see the preceding section). The last two arguments, *Fv* and *Type*, are optional. If you omit the last one or both of them, Quattro Pro assumes their values are zero. See "Financial functions" on page 13 for more information on how @PVAL and @PV relate.

Examples Your grandfather remembers you in his will and leaves you $24,000 in cash over the next 12 years ($2000 a year) *or* you can have all his government bonds, which mature in 15 years to a worth of $30,000. (Whichever you don't choose goes to charity.) To determine which is worth more, compute the present value of the $24,000. Assume you can invest the money as you accumulate it in a 10% money market account.

`@PVAL(10%,12,2000,0,0)` = –13,627.38

The result is negative because the money you invest is considered an outgoing cash flow. Now compare this figure with the present value of the $30,000, which you won't receive for 15 years:

`@PVAL(10%,15,0,30000,0)` = –7,181.76

These results tell you that the $24,000 spread over 12 years is a sounder investment than the other choice, which is presently worth only $7000.

@RADIANS

Format @RADIANS(*X*)

X = a numeric value representing degrees

@RADIANS converts the given number of degrees to radians, using this formula:

$$\frac{\pi}{180} X$$

One degree is equal to approximately 0.017 radians.

Examples `@RADIANS (1)` = 0.017453
`@RADIANS (57)` = 0.994838
`@RADIANS (@DEGREES (3.5))` = 3.5
`@RADIANS (A4)` = 0.994838 (where cell A4 contains the value 57)

@RAND

Format @RAND

@RAND returns a fractional random number between 0 and 1. This offers a sampling of figures, useful for generating sample data for simulated situations.

To generate random numbers in another range, multiply @RAND by the difference between the new high and low ends, then add the new low end number. The formula is @RAND * (high number – low number) + low number.

For example, to indicate a range of 10 to 100, enter @RAND*90+10. This extends the upper limit to 100 and the lower limit to 10.

Note @RAND generates a new random number with each recalculation.

Examples `@RAND` = a random number between 0 and 1
`@RAND*9+1` = a random number between 1 and 10
`@RAND*1000` = a random number between 0 and 1000
`@RAND+5` = a random number between 5 and 6
`-@INT(@RAND*90+10)` = a random integer between –10 and –100

@RATE

Format @RATE(*Fv*,*Pv*,*Nper*)

Fv = a numeric value representing the future value of an investment
Pv = a numeric value representing the current value of an investment
Nper = a numeric value > 0 representing the length of the investment in terms of the number of compounding periods

@RATE calculates the interest rate required in order for an investment of *Pv* to be worth *Fv* within *Nper* compounding periods. If *Nper* represents years, an annual interest rate results; if *Nper* represents months, a monthly interest rate results, and so on.

@RATE uses this formula to calculate interest rate:

$$\left(\frac{Fv}{Pv}\right)^{1/Nper} - 1$$

An equivalent for this formula using @IRATE (see page 61) is

@IRATE(*Nper*,0, –*Pv*, *Fv*)

@RATE assumes that the investment is an ordinary annuity. @IRATE, which is calculated differently than but is related to @RATE, lets you use an optional argument (*Type*) to indicate whether the investment is an ordinary annuity or an annuity due. "Financial functions" on page 13 describes the relationship between @RATE and @IRATE.

Examples To determine what yearly interest rate would double an initial investment of $2000 at the end of 10 years, you could use this @function:

`@RATE(4000,2000,10)` = 7.18%

Other examples:

`@RATE(10000,7000,6*12)` = 0.50% (monthly)
`@RATE(1200,1000,3)` = 6.27% (yearly)
`@RATE(500,100,25)` = 6.65% (yearly)

@REPEAT

Format @REPEAT(*String*,*Num*)

String = a string value
Num = a numeric value ≥ 0

@REPEAT returns *Num* copies of *String* as one continuous label. This @function is similar to the repeating label prefix (\) in that it repeats one or more characters. The difference is that you can specify exactly how many times you want the string to be repeated. The \ label prefix adjusts the display to fill the column, even when the width is changed. @REPEAT displays a fixed number of copies of *String* and does not change.

When you specify a text string with @REPEAT, it must be enclosed by double quotes.

Examples `@REPEAT("*",10)` = **********
`@REPEAT("good day!",3)` = good day!good day!good day!
`@REPEAT(A5,5)` = the contents of the string in A5 repeated 5 times
`@REPEAT("+",@CELL("width",A1..A1))` = ++++++++++++ if column A is 12 characters wide. If you change the column width, you can press *F9* to adjust the repeat string to fill the cell.
`@REPEAT("=",@LENGTH(A3))` where cell A3 is the last or longest label in a column

@REPLACE

Format @REPLACE(*String*,*StartNum*,*Num*,*NewString*)

String = a string value, representing the text to operate on
StartNum = a numeric value ≥ 0, representing the character position to begin with
Num = a numeric value ≥ 0, representing the number of characters to delete
NewString = a string value, representing the characters to insert at position *Num*

@REPLACE lets you replace characters in a label with a new string. It searches through the given *String* from left to right beginning with the first character (character 0) until it reaches the given character position *StartNum*. Then it removes *Num* number of characters from the string, replacing them with *NewString*.

Both *String* and *NewString* can be either cell references or text strings. If text strings, they must be enclosed by double quotes.

To replace one string with another, specify 0 as *StartNum*. For *Num*, enter a number equal to or greater than the number of characters in *String*.

To insert one string into another string, specify 0 as *Num*.

To add one string to the end of another, specify as *StartNum* a number one greater than the number of characters in *String*.

To delete part or all of a string, specify "" as *NewString*.

The usefulness of @REPLACE is greatly increased when it is used in conjunction with other string functions. For instance, to replace one word with another within a sentence, you can use @FIND and @LENGTH to simplify the search-and-replace operation. For example, this formula searches through A7 for *man*, then replaces *man* with *person*:

```
@REPLACE(A7,@FIND("man",A7,0),@LENGTH("man"),"person")
```

Examples

`@REPLACE("McDougal Corp.",2,6,"Connel")` = McConnel Corp.

`@REPLACE("Leslie J. Cooper",7,3,"")` = Leslie Cooper

`@REPLACE("Sales Salaries",6,0,"Reps'  ")` = Sales Reps' Salaries (There must be a space between Reps and the final quote mark.)

`@REPLACE("355 Howard",11,0," St.")` = 355 Howard St. (There must be a space between " and St.)

@RIGHT

Format @RIGHT(*String*,*Num*)

String = a string value
Num = a numeric value ≥ 0

@RIGHT returns *Num* characters of *String* counting from right to left. It allows you to extract a specified number of characters from the right side of a string or label.

If *String* is a not a valid string, @RIGHT returns ERR. If *Num* is 0, the result is "", or an empty string. If *Num* is greater than or equal to the number of characters in *String*, the entire string is returned.

Examples
`@RIGHT("Jennifer Meyer",5)` = Meyer
`@RIGHT("Jennifer Meyer",25)` = Jennifer Meyer
`@RIGHT("Jennifer   ",6)` = fer (including 3 subsequent spaces)
`@RIGHT("155",1)` = 5
`@RIGHT(123,1)` = ERR (123 is a value)
`@RIGHT(A10,5)` = the last five characters of A10
`@RIGHT(A16,@LENGTH(A16) - @FIND("Roosevelt",A16,0))` = Roosevelt (if A16 = Theodore Roosevelt)

@ROUND

Format @ROUND(*X*,*Num*)

X = a numeric value
Num = a numeric value between –15 and 15

@ROUND adjusts the precision of *X* to *Num* decimal points. *Num* specifies the power of 10 to which *X* is rounded. If *Num* is positive, *X* is rounded *Num* digits to the *right* of the decimal point. If *Num* is negative, *X* is rounded *Num* digits to the *left* of the decimal point. For example, if *Num* is –3, *X* is rounded to the nearest thousand.

If *Num* is 0, *X* is rounded to an integer. If *Num* is not an integer, it is rounded off to the nearest integer.

Examples
`@ROUND(12345.54321,0)` = 12346
`@ROUND(12345.54321,2)` = 12345.54
`@ROUND(12345.54321,-2)` = 12300

@ROWS

Format @ROWS(*Block*)

Block = a block value

@ROWS returns the number of rows within the given block.

Examples `@ROWS(A1..A1)` = 1
`@ROWS(A1..C15)` = 15
`@ROWS(B100..B8192)` = 8093
`@ROWS(NAME)` = 30 (if the block NAME contains 30 rows)

@S

Format @S(*Block*)

Block = a block value

If you enter a single cell address instead of a block, Quattro Pro changes it to a one-cell block (such as C3..C3). Quattro Pro also automatically transforms cells prefixed with an exclamation point (as used in 1-2-3) to a one-cell range (!C3 changes to C3..C3).

@S inspects *Block* and returns the string value of the upper left cell. If that cell contains a numeric or date value or is blank, it returns "" (an empty string).

@S is often used to avoid unnecessary ERR values in the notebook. For example, suppose you want to combine labels in cells A1 and A2, and you enter +A1&A2 in cell A5. If cell A1 contains a value and cell A2 contains a string, the result in cell A5 will be ERR. However, if you use @S, as in @S(A1)&@S(A2), this will guarantee that ERR will not appear in A5.

Examples These examples refer to cells in the notebook.

`@S(A1..A6)` = COMPANY
`@S(A2..A2)` = ABC Inc.
`@S(C2..C4)` = (blank)
`@S(B1..B1)&": "&@S(B2..B2)` = SALES REP: Jones
`@S(B3)&@S(C3)` = Marcus

Sales performance list

	A	B	C	D	E
1	COMPANY	SALES REP	SALES	COMMISSION	
2	ABC Inc.	Jones	$123,630	$3,115	
3	Rogers Co.	Marcus	$160,330	$4,040	
4	Klein Sales	Wong	$145,330	$3,662	
5					

@SECOND

Format @SECOND(*DateTimeNumber*)

DateTimeNumber = a number between –109,571 (January 1, 1600) and 474,816.9999999 (December 31, 3199)

@SECOND returns the second portion of *DateTimeNumber*. *DateTimeNumber* must be a valid date/time serial number. Because only the decimal portion of a serial number pertains to time, the integer portion of the number is disregarded. The result is between 0 and 59.

To extract the second portion of a string that is in time format (instead of serial format), use @TIMEVALUE with @SECOND to translate the time into a serial number (see page 99). You can also use @TIME to enter a time value instead of a serial number (see page 99).

Examples

`@SECOND(.3655445)` = 23
`@SECOND(.2543222)` = 13
`@SECOND(35)` = 0
`@SECOND(@TIME(3,15,22))` = 22
`@SECOND(@TIMEVALUE("10:08:45 am"))` = 45
`@SECOND(@TIMEVALUE("10:08 am"))` = 0

@SHEETS

Format @SHEETS(Block)

Block = a block value, either coordinates or name

@SHEETS returns the number of pages within the given block. This is most often used with 3-D blocks. @SHEETS always returns 1 for 2-D blocks.

Examples

`@SHEETS(B:A1..D:IV1)` = 3
`@SHEETS(A1..C7)` = 1
`@SHEETS(A..E:NAME)` = 5

@SIN

Format @SIN(*X*)

X = a numeric value

@SIN returns the sine of the angle *X*. *X* must be given in radians, not degrees. To convert degrees to radians, use @RADIANS (page 85).

Examples

`@SIN(@RADIANS(30))` = 0.5
`@SIN(@PI/6)` = 0.5

@SLN

Format @SLN(*Cost*,*Salvage*,*Life*)

Cost = a numeric value representing the amount paid for an asset
Salvage = a numeric value representing the value of an asset at the end of its useful life
Life = a numeric value representing the number of years of useful life for the asset

@SLN calculates the straight-line depreciation allowance for an asset over one period of its life, using this formula:

$$\frac{Cost - Salvage}{Life}$$

To compute accelerated depreciation (allowing higher depreciation values in the first years of the asset's life), use @SYD (page 96). To calculate depreciation using the double-declining balance method, use @DDB (page 39).

Examples Assume you just bought a new $4000 computer. Your dealer says you can sell it back to the store for $350 after eight years, but that no one would want to buy it after that. In other words, the *Salvage* value of that computer is $350 and its *Life* is 8. To determine the depreciation allowance of the computer for each year of its life, enter this formula:

`@SLN(4000,350,8)` = 456.25

Other examples:

`@SLN(15000,3000,10)` = $1,200
`@SLN(5000,500,5)` = $900
`@SLN(1800,0,3)` = $600

@SQRT

Format @SQRT(X)

X = a numeric value ≥ 0

@SQRT returns the square root of *X*. If *X* is a negative value, the result of @SQRT is ERR. If *X* is a string or reference to a cell containing a label, the @function returns 0.

Examples `@SQRT(9)` = 3
`@SQRT(2)` = 1.414213562
`@SQRT(144)` = 12
`@SQRT(@SQRT(16))` = 2
`@SQRT(-4)` = ERR

@STD

Format @STD(*List*)

List = one or more numeric or block values

@STD returns the population standard deviation (the square root of the population variance) of all values in *List*. @STDS (page 94) computes the standard deviation of sample data. (See page 11 for an explanation of the difference between population statistics and sample statistics.)

List can be any combination of single cell references, blocks, and numeric values. When more than one component are used, all components must be separated by commas. @STD treats any labels within a block as zero and ignores any blank cells. If the *List* contains only blank cells, however, @STD returns ERR.

@STD determines how much individual values in *List* differ from the mean (average) of all values in *List*. It can be used to verify the reliability of the average; the lower the value returned by @STD, the less individual values vary from the average.

Examples `@STD(48,50,52)` = 1.63
`@STDS(48,50,52)` = 2.00

These examples use @STD to calculate the standard deviation of various sales revenue figures shown in the notebook.

`@STD(B5..E5)` = $7.33
`@STD(A3..E3,260)` = $290.15
`@STD(B3..E8)` = $271.78
`@STD(A15..E20)` = ERR (if the entire block is blank)

Monthly sales report

	A	B	C	D	E	F	G
1							
2		JANUARY	FEBRUARY	MARCH	APRIL		
3	JA	$652	$833	$599	$734		
4	MH	$456	$305	$522	$478		
5	RB	$68	$59	$73	$79		
6	PD	$379	$379	$379	$379		
7	AH	$80	$80	$80	$80		
8	MS	$750	$750	$750	$750		
9							
10	TOTAL	$2,385	$2,406	$2,403	$2,500		
11							

@STDS

Format @STDS(*List*)

List=one or more numeric or block values

@STDS returns the sample standard deviation (the square root of the sample variance) of all values in *List*. @STD computes population standard deviation. (See page 11 for an explanation of the difference between sample and population statistics.)

Refer to the @STD entry for more details and examples.

@STRING

Format @STRING(*X*,*DecPlaces*)

X = a numeric value
DecPlaces = a numeric value between 0 and 15

@STRING converts *X* to a string, rounding *X* to the decimal precision indicated by *DecPlaces*.

Once a number or date has been converted to a label using @STRING, no display formatting can be done to it. To format strings derived from numbers as anything other than General format, you must build a macro that uses the {CONTENTS} keyword (see page 176).

Examples

`@STRING(3.59,0)` = 4
`@STRING(98.6,2)` = 98.60
`@STRING(0.3902,0)` = 0
`@STRING("Harry",0)` = 0
`@STRING(A1,2)` = 10.00 (where A1 = 10)

@SUM

Format @SUM(*List*)

List = one or more numeric or block values

Caution: *Any dates in the block will be converted to serial numbers and included in the calculation. Since this will throw off your sum, avoid including dates in the @SUM argument block.*

@SUM returns the total of all numeric values in *List*. *List* can be any combination of single cell references, blocks, and numeric values. When more than one component is used, they must be separated by commas. Any labels or blank cells within a block are ignored by @SUM.

Instead of using @SUM in a formula, you can often use the SpeedSum™ button on the SpeedBar as a shortcut. It totals the values in any block of cells; the totals appear in empty cells within the selected block.

Examples

`@SUM(101,100,99)` = 300

`@SUM(TOTAL,MONTHLY)` = $15,982 if TOTAL = $7,500 and MONTHLY = $8,482

`@SUM([TEST]A1,[SAMPLE]B:D3)` = 10,500 if cell A1 of page 1 of notebook TEST = 10,000 and cell D3 of page B of notebook SAMPLE = 500

These examples refer to cells in the next figure.

`@SUM(B5..E5)` = $279

`@SUM(A3..E3,260)` = $3,078

`@SUM(A5,534)` = 534

`@SUM(B4..E4,B8..E8)` = $4,761

`@SUM(B3..E8)` = $9,694

Monthly sales report

	A	B	C	D	E	F	G
1							
2		JANUARY	FEBRUARY	MARCH	APRIL		
3	JA	$652	$833	$599	$734		
4	MH	$456	$305	$522	$478		
5	RB	$68	$59	$73	$79		
6	PD	$379	$379	$379	$379		
7	AH	$80	$80	$80	$80		
8	MS	$750	$750	$750	$750		
9							
10	TOTAL	$2,385	$2,406	$2,403	$2,500		
11							

@SUMPRODUCT

Format @SUMPRODUCT(*Block1*,*Block2*)

@SUMPRODUCT(*Block1*, *Block2*) returns the dot product of the vectors corresponding to the blocks. Quattro Pro multiplies each corresponding cell from *Block1* and *Block2* and then totals those results.

The blocks must be the same size (same number of rows and same number of columns), or else the blocks must both be one-dimensional (either a row or a column) and they must have the same length. If the blocks don't match, @SUMPRODUCT returns ERR.

Examples Assume the following values for these cells:

A1 = 1	B1 = 5
A2 = 2	B2 = 6
A3 = 3	B3 = 7
A4 = 4	B4 = 8

`@SUMPRODUCT(A1..A2,B1..B2)` = 17 (because 1*5 + 2*6 = 5+12 = 17)
`@SUMPRODUCT(A1..A4,B1..B4)` = 70
`@SUMPRODUCT(A1..A4,B1..B5)` = ERR (blocks are not the same size)

@SYD

Format @SYD(*Cost*,*Salvage*,*Life*,*Period*)

Cost = a numeric value representing the initial cost of an asset
Salvage = a numeric value representing the value of the asset at the end of its life expectancy
Life = a numeric value representing the length of the asset's life expectancy
Period = a numeric value representing the period for which you want to calculate depreciation

This must be true:

$Cost \geq Salvage \geq 0$
$Life \geq Period \geq 1$

@SYD calculates depreciation amounts for an asset using an accelerated depreciation method. This allows higher depreciation

in the earlier years of the asset's life. @SYD uses this formula to compute depreciation:

$$\frac{(Cost - Salvage)\ (Life - Period + 1)}{Life\ (Life + 1)/2}$$

Examples Assume you just bought a new $4000 computer. Your dealer says that you can sell it back to the store for $350 after eight years, but that no one would want to buy it after that. In other words, the *Salvage* value of that computer is $350 and its *Life* is 8. To see what the depreciation allowance of this computer will be by the second year (using this method of depreciation), enter this formula:

`@SYD(4000,350,8,2)` = 709.72

These examples show depreciation values for the first five years of an asset's life. These can be compared to those calculated with @DDB, which distributes more of the depreciation in the first year of life.

`@SYD(12000,1000,5,1)` = $3,667
`@SYD(12000,1000,5,2)` = $2,933
`@SYD(12000,1000,5,3)` = $2,200
`@SYD(12000,1000,5,4)` = $1,467
`@SYD(12000,1000,5,5)` = $733

`@DDB(12000,1000,5,1)` = $4,800
`@DDB(12000,1000,5,2)` = $2,880
`@DDB(12000,1000,5,3)` = $1,728
`@DDB(12000,1000,5,4)` = $1,037
`@DDB(12000,1000,5,5)` = $555

@TAN

Format @TAN(*X*)

X = a numeric value

@TAN returns the tangent of the angle *X*. *X* must be given in radians, not degrees. To convert degrees to radians, use @RADIANS (page 85).

Examples `@TAN(4)` = 1.157821
`@TAN(@PI/4)` = 1
`@TAN(@RADIANS(45))` = 1

@TERM

Format @TERM(*Pmt*,*Rate*,*Fv*)

Pmt = a numeric value representing the amount of the periodic payment
Rate = a numeric value representing a fixed, periodic interest rate accrued by the investment
Fv = a numeric value representing the future value of the investment

@TERM computes the number of payment periods required in order to accumulate an investment of *Fv*, making regular payments of *Pmt* and accruing interest at the rate of *Rate*.

@TERM uses this formula:

$$\frac{\ln(1 + Fv/Pmt\ Rate)}{\ln(1 + Rate)}$$

An equivalent for this formula using @NPER (see page 75) is

@NPER(*Rate*, – *Pmt*, 0, *Fv*)

@TERM assumes the investment is an ordinary annuity. @NPER, which is calculated differently than but is related to @TERM, lets you use an optional argument (*Type*) to indicate whether the investment is an ordinary annuity or an annuity due. "Financial functions" on page 13 describes the relationship between @TERM and @NPER.

Examples To determine how long it will take to accrue $50,000 by depositing $2000 at the end of each year into a savings account that earns 11% annually, enter this formula:

`@TERM(2000,11%,50000)` = 12.67

Quattro Pro determines that it will take 12.67 years to accumulate $50,000 in your account (depending on how your bank pays interest, your balance might not exceed $50,000 until the end of the 13th year).

If, on the other hand, the money is not coming in to you but is being paid out by you, you can enter the future value as a negative number.

You can also use @NPER to calculate this example:

`@NPER(11%,-2000,0,50000,0)` = 12.67

Other examples:

```
@TERM(300,6%,5000) = 11.9 years
@TERM(500,7%,1000) = 1.94 years
@TERM(500,.07,1000) = 1.94 years
@TERM(1000,10%,50000) = 18.8 years
@TERM(100,5%,1000) = 8.3 years
```

@TIME

Format @TIME(*Hr,Min,Sec*)

Hr = a number between 0 and 23, representing Hour
Min = a number between 0 and 59, representing Minute
Sec = a number between 0 and 59, representing Second

@TIME returns the date/time serial number represented by *Hr:Min:Sec*. Each of these arguments must be within the valid ranges above. Any fractional portions are truncated. You can display the resulting time string values in standard time formats by choosing Numeric Format in the block Object Inspector.

Examples `@TIME(3,0,0)` = 0.125 (3:00 am)
`@TIME(3,30,15)` = 0.14600694444 (3:30:15 am)
`@TIME(18,15,59)` = 0.76109953704 (6:15:59 pm)
`@TIME(B15,23,45)` = 0.099826388889 (when the value in B15 is 2)
`@TIME(@HOUR(C3),A4,B10)` = 0.5751388889 (1:48:12 pm) (when C3 = 01:23:13 pm (Formatted date/time serial number), A4 = 48, and B10 =12)

@TIMEVALUE

Format @TIMEVALUE(*TimeString*)

TimeString = a numeric value or a string value in any valid time format, enclosed by quotes

@TIMEVALUE returns a serial time value that corresponds to the value in *TimeString*. If the value in *TimeString* is not in the correct format, or is not enclosed in quotes (if entered as a literal string), an ERR value is returned.

You can display resulting time string values in standard time formats by choosing Numeric Format in the block Object Inspector.

There are four valid formats for *TimeString*:

- HH:MM:SS AM/PM (03:45:30 PM)
- HH:MM AM/PM (03:45 PM)
- The Long International Time Format chosen as a system default, one of which is HH:MM:SS (15:45:30)
- The Short International Time Format chosen as a system default, one of which is HH:MM (15:45)

Examples

`@TIMEVALUE("03:30:15 AM")` = 0.1460069444
`@TIMEVALUE("03:00")` = 0.125
`@TIMEVALUE("18:15:59")` = 0.76109953704
`@TIMEVALUE("3.45")` = ERR
`@TIMEVALUE(@TIME(12,30,45))` = 0.521354
`@TIMEVALUE(A1)` = 0.125 if A1 contains the label '03:00

@TODAY

Format @TODAY

@TODAY enters the numeric value of the system's date. It is equal to the expression @INT(@NOW).

@TRIM

Format @TRIM(*String*)

String = a string value

@TRIM removes any extra spaces from *String*: that is, spaces following the last non-space character or preceding the first non-space character, and duplicate spaces between words. Strings with no extra spaces are not affected. If *String* is empty or contains a numeric value, it returns ERR.

Examples

`@TRIM(" too     many     spaces    ")` = too many spaces
`@TRIM("no extra spaces")` = no extra spaces
`@TRIM(125)` = ERR

@TRUE

Format @TRUE

@TRUE returns the logical value 1 and is usually used in @IF formulas. The 1 it returns is the same as the regular numeral 1, but @TRUE makes the formula easier to read.

See also @FALSE on page 50.

Examples
`@TRUE` = 1
`@IF(C3=100,@TRUE,10)` = 1 (if C3 = 100) or 10 (if C3 ≠ 100)
`@IF(C3=100,@TRUE,@FALSE)` = 1 (if C3 = 100) or 0 (if C3 ≠ 100)

@UPPER

Format @UPPER(*String*)

String = a string value

@UPPER returns *String* in uppercase characters. Numbers and symbols within a string are unaffected. If *String* is blank, or contains a numeric or date value, the result is ERR.

Examples
`@UPPER(4839)` = ERR
`@UPPER(@LEFT("johnson",1))` = J
`@UPPER("upper")` = UPPER
`@UPPER("Hello, world.")` = HELLO, WORLD.
`@UPPER("145 Bancroft Lane")` = 145 BANCROFT LANE

@VALUE

Format @VALUE(*String*)

String = a string value

@VALUE converts *String* into a numeric value. *String* can contain arithmetic operators (but don't place arithmetic operators within quotes). *String* must not contain embedded spaces. Dollar signs, commas, and leading and trailing spaces are ignored.

This @function is useful for converting data that was imported with Tools | Import, but was not automatically converted into values.

Examples `@VALUE("   3.59")` = 3.59 (leading spaces are stripped)
`@VALUE("   98.6   ")` = 98.6 (leading and trailing spaces are stripped)
`@VALUE("98.6 4")` = ERR (an embedded space is not allowed)
`@VALUE(3+4)` = 7
`@VALUE("3+4")` = ERR (arithmetic operators within quotes are not allowed)
`@VALUE(" 88.039")` = 88.039
`@VALUE(A10)` = 56.34 (where cell A10 = '$56.34)
`@VALUE("34,200")` = 34200
`@VALUE(A1)` = ERR (where cell A1 = '1800 Green Hills Road)

@VAR

Format @VAR(*List*)

List = a list of values

@VAR calculates the population variance of all non-blank, numeric cells in *List*, using the *n* method (biased). Refer to the formulas on page 11 to see how @VAR computes the variance of population data.

See page 11 for an explanatory note about the difference between population statistics and sample statistics. @VARS (see the next entry) computes variance with sample data, which is more general and useful.

Caution! If *List* contains text, a reference to a single cell containing a label (for example, `@VAR(B1)` where B1 = Adam), or label cells within references to multiple cell blocks (such as `@VAR(B1..B5)`), @VAR treats the string as having a value of 0. @VAR ignores blank cells within a referenced block of cells, but returns ERR if every cell in the block is blank.

Examples `@VAR(48,50,52)` = 2.67
`@VARS(48,50,52)` = 4.00
`@VAR("Adam",53)` = 702.25—same as for @VAR(0,53)
`@VAR(B1..B4)` = 54.6875 (if B1=10, B2=15, B3="Susan", B4=20; the string in B3 is treated as if it were 0)

@VARS

Format @VARS(*List*)

List = a list of values

@VARS calculates the sample variance of all non-blank, numeric cells in *List*, using the *n-1* method (unbiased). Refer to the formulas on page 11 to see how @VARS computes the variance of sample data.

@VARS calculates variance with sample data. @VAR (the previous entry) computes variance with population data. See page 11 for an explanation of the difference between sample statistics and population statistics.

Refer to the @VAR entry for details and examples of how to use @VARS.

@VERSION

Format @VERSION

@VERSION returns the version number of Quattro Pro for Windows.

Example @VERSION returns 101 for Quattro Pro for Windows version 1.

@VLOOKUP

Format @VLOOKUP(*X*,*Block*,*Column*)

X = a string or numeric value
Block = a 2-D block value
Column = a numeric value ≥ 0 and < the number of columns in *Block*; the first column has a value of 0

@VLOOKUP works like @HLOOKUP, except that rows and columns are reversed.

@VLOOKUP searches (vertically) down the first column of the given *Block* for value *X*. Then, it counts over the specified number of columns (indicated by *Column*) and returns the value found in that cell. The first column has a value of 0; the second has a value of 1, and so on.

The value of *X* can be either a character string or a number, the address or block name of any cell containing a label or value, or any expression that results in a number or string. If *X* is a string, the match must be exact; the lookup is case-sensitive. If *X* is a number and @VLOOKUP can't find an equal number, it locates the highest number in the column not greater than *X*.

The second argument (*Block*) specifies the coordinates of the table to be used for the lookup. This table must have its *index* values in the leftmost column. These values (if numbers) must be in ascending numerical order. Also, there must be no blank cells in the index column. Blanks in the table to the right of the index column are treated as 0.

@VLOOKUP looks through the index column of the table from top to bottom looking for a match to *X*. If it finds an exact match, the lookup stops at that column. If a match isn't found and *X* is a number, the search stops at the value closest to but not greater than *X*. If *X* is a number and the index column contains only labels, or if *X* is a label and an exact match is not found, @VLOOKUP returns ERR.

The final argument (*Column*) tells @VLOOKUP how many columns to the right of the index column the return value is. This argument can be any value from 0 up to the number of columns in the table, or any expression or cell address resulting in such a value. A value of 0 tells @VLOOKUP to return the actual value found in the index column. A value of 1 instructs @VLOOKUP to return the value directly to the right of the one found in the index column; a value of 2 tells @VLOOKUP to return the value two columns to the right, and so on.

Note If *X* is a string value and *Column* = 0, @VLOOKUP returns the offset number of the row *X* is found in, not the value of *X*.

These instances result in ERR:

- *Column* is less than 0 or greater than the number of columns in *Block.*
- *X* is less than the smallest value in the first column of *Block.*
- *X* is a string and an exact match in the index column is not found.

Examples These examples refer to data in the lookup table. In the first example, @VLOOKUP searches down the first column of the specified block (column A), looking for the largest number equal to or less than 17. It stops at cell A3, then moves across the

specified number of columns (3). It stops at cell D3 and returns the value 22.

```
@VLOOKUP(17,A1..G7,3) = 22
@VLOOKUP(10,A1..G7,0) = 10
@VLOOKUP(6,A1..G7,2) = 84
@VLOOKUP(50,A1..G7,3) = 62
@VLOOKUP("string",A1..G7,3) = ERR (no labels in block)
@VLOOKUP("18",A1..G7,2) = ERR (no labels in block)
@VLOOKUP(18,A1..G7,8) = ERR (col value > # cols)
@VLOOKUP(18,A1..C7,4) = ERR (col value > # cols in given block)
```

Vertical lookup table

	A	B	C	D	E	F	G	H
1	5	52	84	43	96	37	38	
2	10	32	67	45	48	90	41	
3	15	42	18	22	32	89	76	
4	20	83	76	47	35	42	43	
5	25	48	82	41	26	25	72	
6	30	85	72	46	12	71	51	
7	35	64	56	62	52	91	33	
8								

To search horizontally through a table, use @HLOOKUP (page 54).

@YEAR

Format @YEAR(*DateTimeNumber*)

DateTimeNumber = a number between –109,571 (January 1, 1600) and 474,816.9999999 (December 31, 3199)

@YEAR returns the year portion of *DateTimeNumber*. To display the actual year, just add 1900 to the result of @YEAR. If you want to extract the year portion of a string that is in date format, use @DATEVALUE with @YEAR to convert the string into a serial number.

Examples

`@YEAR(22222)` = 60 (1960)
`@YEAR(A6)+1900` = 19*nn*, where *nn* is the year value in A6
`@YEAR(@DATEVALUE("12-Oct-54"))` = 54
`@YEAR(-75000)` = –206 (1694)

C H A P T E R

3

Using macros

Quattro Pro's macro facility offers commands to automate user tasks and design applications. This chapter describes how to

- record macros
- store macros in a separate notebook
- run macros
- type macros in the notebook without recording them
- debug macros using the debugger utility

What is a macro?

A *macro* is a sequence of commands Quattro Pro runs automatically. When running a macro, Quattro Pro performs the actions listed in it. You can store macros in the notebook they apply to or in a macro library file for use by other notebooks.

Macros can reproduce the behavior of keys on the keyboard, mouse actions, and menu commands. Special macro commands can perform actions such as prompting the user for input, looping a macro repeatedly, or controlling other Windows applications. Use them to automate complex or repetitive command sequences (like printing a standard report), to enter frequently used labels with a keystroke, or to build complete applications for use by people with little Quattro Pro experience.

Recording macros

With Quattro Pro, you can *record* actions as you perform them. Recording converts actions into *macro commands* and stores them as labels in a block.

Basic macro recording

Think about the quickest way to perform the task at hand. Longer recording times mean more chances for error. To record a macro,

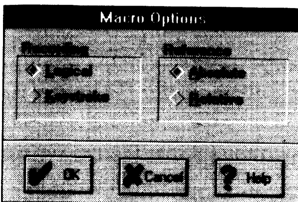
1. Choose Tools | Macro | Record.
2. Select the block to store the macro in. If there's no chance of overwriting data in cells below, select a single cell; Quattro Pro fills the cells below as necessary. Selecting a block larger than a cell confines recorded commands to that block only, and the macro stops recording when the block is full.

   Caution: Make sure the task being recorded won't affect the cells you're recording macro commands into; record macros into a column that the task doesn't need.
3. Choose OK to place Quattro Pro in recording mode (noted by the REC indicator on the status line).
4. Perform your task. Quattro Pro records each action as you perform it, writing it as one or more macro commands in the first column of the block previously specified. You don't have to keep this block visible while recording.
5. Choose Tools | Macro | Stop Record to finish recording. The macro is now stored in the block.
6. Move to the first cell of the macro, and choose Block | Names | Create. Enter a name for the macro and choose OK. See page 117 for more information on naming macros.

Once you record a macro, you can run it by choosing Tools | Macro | Execute, entering the macro's name (or the cell address of its first command), and choosing OK. Running macros is discussed in more detail on page 111.

Recording modes

By default, when you look at a recorded macro you can see Quattro Pro doesn't record actions keystroke by keystroke. Instead, it translates keystrokes and mouse actions that invoke menu choices into one or more macro commands called *command equivalents*. (For a complete discussion of command equivalents, see page 129.) This recording method is called *logical* recording. Macros recorded this way are more efficient and can run regardless of the current Quattro Pro menu system. For example, a macro containing commands to print the current notebook will run even if the File | Print command is not in the active menu bar.



To record keystroke macros, choose Keystroke from the Tools | Macro | Options dialog box. Macros recorded in keystroke mode use keystroke commands to select menu commands. For example, the keystroke macro for copying A1 to A2..A46 is

{ALT+B}C{ALT+F}A1{ALT+T}A2..A46{CR}

Using keystroke macros to emulate menu operations isn't recommended. The previous example shows many of the disadvantages of using keystroke commands:

- You can't easily tell what the macro does by reading it.
- The macro works only in menu systems with Block | Copy, and it works incorrectly in menu systems with a command whose menu shortcut key is B and shortcut key is C (for example, Block | Combine).
- Because each keystroke must be processed as a separate macro command, the macro runs more slowly.

Note

- When recording in Keystroke mode, mouse actions are ignored.
- Keystroke macros might not run in future releases of Quattro Pro.

See page 129 for a discussion of command equivalents, which let you perform menu operations that aren't dependent on the menu system in use.

If Quattro Pro records your menu actions keystroke by keystroke instead of as command equivalents, choose Logical from the Options dialog box to revert to logical recording.

Standard addresses vs. relative references

By default, addresses in macro commands use standard cell addressing like A1..C6. By choosing Relative from the Tools | Macro | Options dialog box, you can record macro commands that use relative references like C(0)R(2). See page 136 for more information on using relative references.

Recording tips

Here are some handy tips to make macro recording easier:

- Use the shortest steps possible to perform your task. The longer a recording session goes, the more errors can result.
- Use any method available to choose menu commands when recording. The resulting macro reads the same when recorded logically.
- You can't record a macro that pauses for user input. If you want the macro to pause, edit it after recording it. (Typing macros is discussed on page 117.)

Macro libraries

The best way to store macros is in a *macro library*, a special notebook reserved for macros. You can use it to store macros you want to access from any notebook. There are many advantages to storing macros in a library:

- It simplifies linked access.
- It's easier to keep track of where the macros are.
- If Quattro Pro can't find the macro you specify within the active notebook, it searches through all open macro libraries until it finds it.
- The macros don't interfere with the notebook data and vice versa.
- Using one set of macros for a group of notebooks saves disk space.
- If you *do* want to insert a set of macros into individual notebooks, you can store them in a library, then copy the macros into notebooks as you need them.

- You can have a separate set of macros for each application or type of notebook you work with.

To create a macro library,

A macro library must be saved in either Quattro Pro for DOS or Quattro Pro for Windows format.

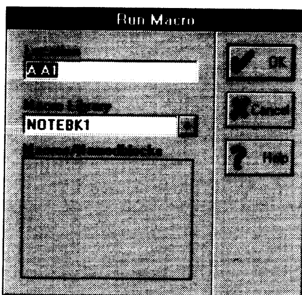
1. Open a new notebook. (You can also use an existing notebook.)
2. Create the macros to store in it (or copy them to it from another notebook). See page 136 for a discussion of how addressing is handled in a macro library.
3. Inspect the notebook and choose Macro Library.
4. Choose Yes to define the active notebook as a macro library.
5. Choose OK to save the change.
6. Choose File | Save to save the macro library.

Note It's a good idea to keep only one macro library open at a time. If two open libraries contain a macro with the same name, it's difficult to predict which macro will run. Quattro Pro searches only open macro libraries for macros; closed libraries are ignored.

Use Window | Hide (or {WindowHide}, its equivalent) to hide your macro libraries. This prevents others from altering them.

Running macros

To run a macro,



1. Choose Tools | Macro | Execute (or press *Alt+F2*).
2. If desired, use Macro Library to specify the name of the notebook containing the macro. This displays a list of macros available under Macros/Namedblocks.
3. Choose the name of the macro to run from Macros/Namedblocks or choose Location and enter the macro's block coordinates. If the macro is in another notebook, use full linking syntax (for example, [LIBRARY]A:C26). Linking isn't needed if entering a block name.
4. Choose OK to run the macro.

The MACRO indicator appears on the status line while a macro runs. If the macro contains an error, Quattro Pro beeps, displays an error message, and stops the macro. You can use the debugger utility (discussed on page 139) to pinpoint and correct macro

errors. When Quattro Pro runs a macro, it begins with the first cell in the macro block. It continues down through the column, interpreting everything it encounters as part of the macro, until it finds

- an empty cell
- a cell containing a value
- the {QUIT} macro command
- the {RETURN} macro command
- an invalid macro entry

At that point, the macro stops and control returns to the user. Leave an empty cell, {QUIT}, or {RETURN} after each macro in a notebook, or the macro will try to interpret labels below the macro as commands.

To stop a macro while it's running, press *Ctrl+Break* and choose OK to clear the break message and return to READY mode.

Note If the *Ctrl+Break* key has been disabled with {BREAKOFF}, you won't be able to interrupt the macro. If you need to use {BREAKOFF} in the macro, add it as a last step after debugging the macro.

Suppressing screen redraw

By default, Quattro Pro doesn't show the actions a macro performs while it's running (this makes the macro run faster). To see the actions a macro performs, right-click the application title bar and choose None from the Macro property's Macro Suppress-Redraw option. For a complete discussion of screen suppression, see the discussions of {PANELOFF} and {WINDOWSOFF} in Chapter 4.

Running macros from Windows

You can run a macro as you load Quattro Pro by specifying its name after you enter *QPW* and a file name from the Program Manager or the DOS command line. For example, the following DOS command brings up Windows, loads Quattro Pro, retrieves the notebook named EXPENSE, and runs the macro named \F:

```
WIN QPW EXPENSE \F
```

Only `QPW EXPENSE \F` is necessary when using the Program Manager's File | Run command. Windows applications that support Dynamic Data Exchange (discussed on page 131) can make Quattro Pro run macro commands; see their documentation for more details.

Attaching a macro to a key

If the macro you're creating is used often, you can give it a special block name that makes it easier to run: name it with a backslash (\) followed by a letter of the alphabet (*A-Z*). (Use the naming methods described on page 117.) Then you can run the macro from the keyboard by pressing *Ctrl* and that letter (either uppercase or lowercase). Because there can be only 26 of these macros (one for each letter of the alphabet), reserve them for macros you use often.

Note Macros assigned to *Alt* keys in the DOS version of Quattro Pro run in the Windows version using the *Ctrl* key.

Attaching macros to notebook buttons

You can use the SpeedButton tool on the notebook SpeedBar to add buttons (called *macro buttons*) that run macros in a notebook. To create a macro button,

1. Choose the SpeedButton tool from the notebook SpeedBar.
2. To create the macro button in a default size, just click. The button appears, with its top left corner at the position you clicked.

   To create the control in a specific size, hold down the left mouse button and drag the pointer. As you drag, a bounding box indicates the size of the button; release the mouse button to create the macro button.
3. There are two ways to change the appearance of the button:

   To select a macro button, right-click it and press Esc; clicking the macro button runs its macro.

   - You can paste a graphic image onto the button: copy a Windows bitmap into the Clipboard, select the macro button, and choose Edit | Paste.
   - You can change the text appearing on the button: right-click it, choose Label Text, and enter the text to display on the button. Choose OK to set the new button title.
4. Right-click the macro button and choose Macro. Enter the macro commands to run when the button is clicked. You can use {BRANCH} to run a macro in a notebook. Choose OK to save the macro.

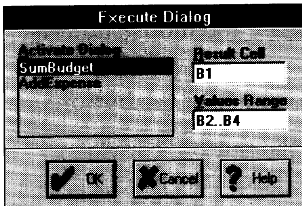
See page 134 and the description of {FLOATCREATE} in Chapter 4 for information on creating macro buttons using macro commands.

Once a macro button is created, you can right-click it to change the properties listed in the following table:

Table 3.1
Macro button properties

Property	Description
Macro	Macro commands to run when the button is clicked. For example, {BEEP}{SHIFT+DOWN 3}. You can use **Execute Dialog** to specify a dialog box to display (discussed next).
Label Text	The text titling the button.
Border Color	The color of the button's border.
Box Type	The thickness of the button's border, and whether it has a shadow.
Object Name	The name of the object. Used in object linking to manipulate its properties.

You can use the Execute Dialog option of the Macro property to make the macro button display a dialog box. Using this option replaces whatever macro is assigned to the button with a {DODIALOG} command. To display a dialog box with a macro button,



1. Right-click the macro button and choose Macro.
2. Click Execute Dialog.
3. Choose the name of the dialog box to display from the Activate Dialog list box.
4. Choose Result Cell and point to a cell in the notebook. After the dialog box is closed, this cell will contain 1 or 0. The value 1 means the dialog box was closed by OK; the value 0 means it was canceled.
5. Choose Values Block and point to a block in the notebook that contains initial settings for the dialog box. When the dialog box is closed, its final settings are written back into this block. See page 329 for more information on the setting block.
6. Choose OK to create the {DODIALOG} command that displays this dialog box.
7. Choose OK to save the macro.

Attaching macros to graph buttons

You can make a text box in a graph run macros (making it into a *graph button*) using its Graph Button property:

1. Right-click the text box.
2. Choose Graph Button and check Execute Macro to specify that this text box will run a macro when clicked. The macro won't run if you click the text box in a graph window; it runs when you view the graph in a slide show or with Graph | View.
3. Enter the macro to run in the edit field provided. For example, {SLIDE.GOTO "Quarter3"}.
4. Choose OK to save the change.

For more information on creating graph buttons, see Chapter 10 of the *User's Guide*.

Assigning macros to dialog box objects and menu commands

Many objects in a dialog window can run macros using Dialog | Links. See Chapter 6 for a complete discussion of object linking and the DOMACRO link command. You can make menu commands that run macros with {ADDMENU} and {ADDMENUITEM}; see Chapter 7 for details.

Autoload macros

By default, any macro named \0 (backslash zero) runs automatically when the notebook containing it opens. Each notebook can have one of these *autoload* macros. To change the name of the autoload macro,

1. Right-click the application title bar.
2. Choose Startup.
3. Choose Startup Macro and enter the autoload macro's new name.
4. Choose OK to save the change.

When the autoload macro is renamed, the old autoload macro name no longer runs automatically in any notebook. For example, if you set the autoload name to `autostart`, macros named `\0` would no longer run automatically; they'd have to be renamed `autostart`.

To ensure that an autoload macro runs in a notebook regardless of whether it's linked to another, begin it with {ESC}. This cancels the dialog box that appears when you open a notebook containing links. You can use Tools | Update Links to access linked values or to load supporting notebooks after accessing the file.

Quattro Pro for DOS macros

If you've written macros using menu-equivalent commands such as {/ Block;Copy} in the DOS version of Quattro Pro, you can run them in Quattro Pro for Windows without modification. (See the Quattro Pro for DOS *@Functions and Macros* guide for more information on menu-equivalent commands. Appendix A lists Quattro Pro for Windows menu-equivalent commands, which are called command equivalents.)

To run Quattro Pro for DOS macros written with keystrokes, such as /EC,

1. Right-click the application title bar.
2. Choose Macro and uncheck Key Reader (if it's checked).
3. Choose Slash Key and select Quattro Pro – DOS.
4. Choose OK to save the change and return to the active window.

This also lets you press the slash key (/) to display a menu system that's keystroke compatible with Quattro Pro for DOS.

Macros assigned to *Alt* keys in the DOS version of Quattro Pro run in the Windows version using the *Ctrl* key.

1-2-3 macros

To run macros written for 1-2-3,

1. Right-click the application title bar.
2. Choose Macro.
3. Check Key Reader.
4. Choose OK to save the setting and return to the active window.

Macros assigned to *Alt* keys in 1-2-3 run in Quattro Pro using the *Ctrl* key.

Typing macros

You can type macros directly into cells as labels instead of recording them. This method requires precision (one incorrect keystroke can invalidate a macro), so it's not recommended for novices. Some macros can only be typed, such as those that prompt for user input or use Dynamic Data Exchange (DDE) to communicate with other Windows applications.

The basic procedure

To type a macro,

Each macro command is discussed in depth in Chapter 4 and Appendix A. For a general discussion of macro commands, along with an overview of each command category, see page 122.

1. Move to the first cell of the block in which you want to store the macro.
2. Enter the macro commands that perform your task. Certain keys, such as function keys, must be indicated with special commands like {EDIT}. You can enter multiple macro commands in the same cell or in the cell below the last command entered. If using more than one cell, be sure the entries proceed downward (A1, A2, A3) with no empty cells between them. Each cell must contain a label or a text formula.
3. Leave an empty cell, {QUIT}, or {RETURN} at the end of the macro. Otherwise the macro interprets labels below it as macro commands.

Block names and macros

When you finish entering a macro, name its first cell using Block | Names | Create. Then you can run the macro by choosing its name from the Tools | Macro | Execute dialog box or pressing a *Ctrl* key shortcut. (See page 111 for a complete discussion of running macros.) Block names make macros more readable and easier to remember. Give your macros names reflecting their task. For example, PERCENT could be the name of a macro that changes the numeric format of the current cell to percentage.

Caution! When naming macros, avoid names that duplicate macro commands. Otherwise the macro commands become invalid. For example, if you name a macro READLN, which is the name of a predefined macro command, and then try to read a line from a file into cell A6 using {READLN A6}, Quattro Pro assumes you're

running the READLN macro instead of running the predefined {READLN} command.

Macro tips

Plan your macro carefully. Making it efficient saves you the trouble of modifying it later to add new functionality. If it involves menu commands, step through them first, writing down each action involved. If it involves special macro commands, know exactly what you're going to enter as the command arguments. Keep track of the selector after each macro command; its position can affect the macro's behavior.

Flow charts

Try sketching a *flow chart* of your macro before entering it. A flow chart is a diagram showing the actions and decisions a macro steps through while running. Boxes in the chart contain an action the macro performs. Diamonds contain decisions the macro must make to determine the next step; for example, a cell achieving a certain value or the selector reaching a specific location.

Arrows show where to go after a step is completed. Arrows exiting a diamond have a decision listed next to them dictating the next step. If a decision isn't listed, the step at the end of the arrow is *always* the next step. Flow charts don't show actual macro commands; they show the planning and logic behind the macro. The next figure is an example of a flow chart.

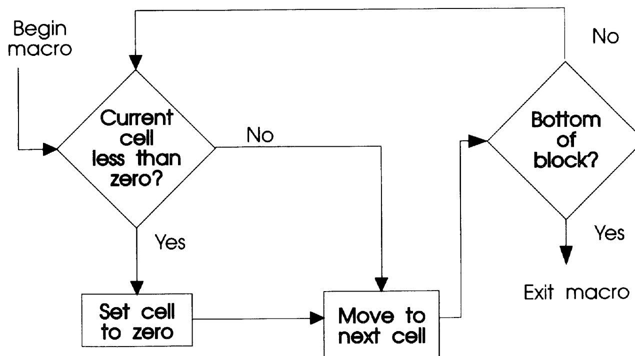


Figure 3.1
A macro flow chart

This figure charts a macro that replaces each negative number in a block with zeros.

Typing tips

Here are some suggestions to follow when typing macros. They'll make your work much easier:

- Use Tools | Macro | Record to create sections of a macro that consist of fixed keystrokes and menu choices. This speeds entry time and reduces errors.
- If your macro uses multiple commands, place each in a separate cell. This makes problems easier to pinpoint.
- Add comments to your macros to make them easier to understand and debug. The following figure shows a macro written into three cells. Descriptions of each step are included in cells to the right. The name of the macro is shown to the left. All macro examples in this chapter use this figure's format.

Figure 3.2
A commented macro

	A	B	C	D	E	F
1						
2	\f	{BLANK part_list}			Erase part_list block	
3		{EditGoto part_list}			Moves to part_list block	
4		{?}~			Pauses for user input	
5		{BEEP}{QUIT}			Beeps and exits the macro	
6						

- Another advantage of listing all block names to the left of the macros or cells they reference is that you can use Block | Names | Labels to create the block names quickly.
- Place macros in an area unaffected by macro commands. For example, don't place macros to the right of a block where rows will be inserted or deleted. The ideal place for a macro depends on what operations it performs. If you want to use a macro with many notebooks, place it in a macro library (see page 110). If only one notebook uses the macro, add a page named Macros to that notebook and place the macro there.

Macro commands

Each macro is a sequence of *macro commands*. These commands include command equivalents (which emulate menu commands) and unique functions you can't do directly in Quattro Pro, such as sounding the computer's speaker, pausing for a fixed length of time, or prompting for input. Chapter 4 describes each macro command in full detail. Appendix A lists the command equivalents available in Quattro Pro.

Macro syntax

Macro commands are like @functions in that they have specific grammatical rules, or *syntax*. The syntax for macros is fairly straightforward:

{COMMANDNAME *Argument1,Argument2,Argument3...*}

Many command equivalents contain a period in their command name.

COMMANDNAME is the exact name of the command. *Arguments* are values providing instructions to the command. Not all macro commands require arguments; when they do, they require a specific type of information. (See the next section for a description of argument types used in macro commands.) Some examples of macro commands are

```
{Query.Criteria_Table B27..B29}
{BlockCopy A1,A2..A37}
{Search.Find "3rd Quarter Profits"}
{BEEP 3}
{GETNUMBER "How old are you?",AGE}
{CONTENTS E15,F15,5}
```

The syntax rules for macro commands are as follows:

- The command must begin and end with braces { }.
- There must be a space between the command name and the first argument. For example, {GETNUMBER"Number?",A1} results in a syntax error; {GETNUMBER "Number?",A1} works correctly.
- Separate multiple arguments with commas. By changing the Punctuation option of the International property in the application Object Inspector, you can use semicolons or periods instead. (See Chapter 16 of the *User's Guide* for more information on punctuation.)
- Arguments must be the correct type; if a string is required, the argument must be a valid character string or a syntax error occurs.
- If an argument contains spaces or punctuation, enclose it in quotes. For example,

  {GETLABEL "Hello, world",A4}

  is allowed, but not

  {GETLABEL Hello, world,A4}
- You must enter the entire macro command in a single cell.

- You can enter the command in any case. (In this manual, all macro commands except for command equivalents are shown in uppercase.)
- You can include more than one macro command in a cell; for example, {BEEP}{GETLABEL "Hello, world",A4}{QUIT}.

Because macro commands are labels, Quattro Pro won't recognize a syntax error when you type it in; an error occurs when you try to run the macro. To save debugging time, pay careful attention to the format of macro commands as you enter them, and record macros whenever possible.

Arguments in macro commands

Arguments in macro commands, like those used with @functions, require specific information to be supplied with the command. There are four types of arguments: numbers, strings, locations, and conditions.

- **Number** requires any numeric value, entered as
  - an actual number (such as 2 or 0.45)
  - a formula resulting in a number (such as A3*15)
  - a cell address or named block containing a numeric value or formula (such as C10, where C10 contains a valid number or formula)
- **String** requires a text string, entered as
  - an actual string in quote marks ("Borland")
  - a reference to a cell or named block containing a label
  - a comma-separated list of property or command equivalent settings, enclosed in quotes ("Currency,2"). If a setting in the list would normally take a single quote, enter two quotes ("Prefix,""Windows Default"",No"). If a setting contains spaces or punctuation, enclose it in two sets of quotes.
  - a formula resulting in a label, such as @UPPER("hello") (See Chapter 2 of the *User's Guide* for a discussion of text formulas. See Chapter 1 for a discussion of string @functions.)

Quote marks aren't required when the string doesn't contain any spaces or punctuation. You must use quote marks around a numeric value to enter it as a string ("416").

- **Location** requires a reference to a cell or a block. The reference can be
  - a block name
  - coordinates for a block containing one or more cells; for example, A1, A1..A4 or A..B:C4..D22

- the relative reference (discussed on page 137) of a block containing one or more cells; for example, []C(0)R(0), []P(-2):C(0)R(22), or []C(0)R(0)..C(3)R(10)
- coordinates for a noncontiguous block, enclosed in parentheses; for example, (A1,B1..B7,C1..C7) or (A1,B:C27..C52)
- a label or text formula (see Chapter 2 of the *User's Guide*) resulting in any of these options; for example, + "A"&"2", which results in A2

- **Condition** requires a logical formula; that is, a formula that can be evaluated as either true or false; for example, +C4 > 500. (See Chapter 1 for more information on logical formulas.)

Some commands accept a combination or choice of argument types. For example, {LET} stores either a label or a number in a cell, depending on the argument type. You can include *argument suffixes* in the command to specify a value or label entry. Using *:string* (or *:s*) assures a label entry, and *:value* (or *:v*) assures a value entry (if the entry is a valid number). For example, the following macro enters the value 7 into new_block:

```
{LET new_block,3+4:value}
```

The following macro enters 3+4 as a *label* in new_block:

```
{LET new_block,3+4:string}
```

Caution! Unlike @functions, block references in macros *are not updated* when the macro is copied. If you move the contents of a cell, or insert or delete a row or column, the macro reference might be wrong. For this reason, use block names whenever possible. Quattro Pro updates block names to reference the correct location.

Entering macro commands

You can enter macro commands using the Macros key (*Shift+F3*). This displays a menu of command categories:

Macros

- **Keyboard** commands emulate the action of various keys on the keyboard.
- **Screen** commands affect the display.
- **Interactive** commands let you create macros that display dialog boxes or pause for the user to enter data from the keyboard.
- **Program Flow** commands let you branch and loop in a macro.
- **Cell** commands affect the data stored in specified cells.

- **File** commands work with data within files other than the active notebook file.
- **/ Commands** are commands used by Quattro Pro for DOS to emulate menu commands. Use these commands only to create macros that must run under Quattro Pro for DOS. See the Quattro Pro for DOS *@Functions and Macros* guide for more information on menu-equivalent commands. Appendix A lists Quattro Pro for Windows menu-equivalent commands, which are called command equivalents.
- **Command Equivalents** are commands that emulate operations you can perform with menus.
- **DDE** commands manipulate other Windows applications.
- **UI Building** commands let you change the menu bar.
- **Object** commands let you create and change the properties of objects.
- **Miscellaneous** commands let you scroll the active window or insert comments into a macro.

When a list of categories or commands is displayed, you can search for an item in the list by pressing *F2* and typing the first few letters of the item's name. Quattro Pro highlights the first item in the list that begins with the text you type.

To insert a macro command,

1. Press *Shift+F3* to display a list of macro command categories.
2. Choose the category containing the desired command.
3. Choose the command to insert.

Quattro Pro inserts the macro command name on the input line.

The remainder of this section provides an explanation and example of each category. For an alphabetical listing of all macro commands, along with examples, see Chapter 4. Many of the examples shown next are stored in the notebook MACROS.WB1 (included with Quattro Pro).

Keyboard

Keyboard commands emulate the keys of the keyboard. You can repeat many of these macro commands by specifying a repeat number as an argument. For example, to move the selector down five cells, use {DOWN 5}.

You can combine key commands for a variety of purposes. For example, the following macro replaces a formula in the active cell with its result and then moves down one cell:

```
{EDIT}{CALC}{CR}{DOWN}
```

The following macro deletes the last three characters of whatever is in the active cell:

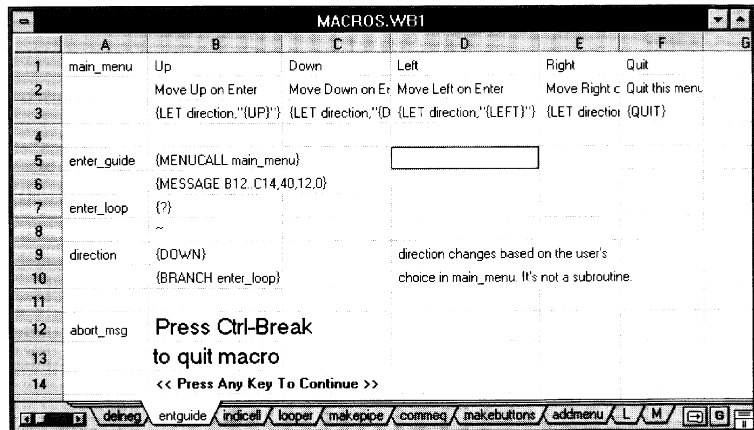
```
{EDIT}{END}{SHIFT+LEFT 3}{DEL}{CR}
```

Screen Screen commands can control the status line, suppress screen redraws, or sound the computer's bell. For example, {INDICATE "WAIT"} changes the status indicator to WAIT; {INDICATE} resets the status indicator to its default.

Interactive Interactive commands control user access to the macro. They can

- display dialog boxes
- display popup menus
- request input
- pause the macro
- prevent the user from breaking out of the macro during a crucial period

The next figure demonstrates a macro that intercepts any occurrence of *Enter* and replaces it with an arrow key (←, →, ↑, ↓) specified by the user.



MACROS.WB1

	A	B	C	D	E	F
1	main_menu	Up	Down	Left	Right	Quit
2		Move Up on Enter	Move Down on Er	Move Left on Enter	Move Right c	Quit this menu
3		{LET direction,"{UP}"}	{LET direction,"{D	{LET direction,"{LEFT}"}	{LET directio	{QUIT}
4						
5	enter_guide	{MENUCALL main_menu}				
6		{MESSAGE B12..C14,40,12,0}				
7	enter_loop	{?}				
8		~				
9	direction	{DOWN}		direction changes based on the user's		
10		{BRANCH enter_loop}		choice in main_menu. It's not a subroutine.		
11						
12	abort_msg	Press Ctrl-Break				
13		to quit macro				
14		<< Press Any Key To Continue >>				

Figure 3.3
An interactive macro

The commands {?}, {MENUCALL}, and {MESSAGE} in this example are interactive commands.

See Chapter 6 for more information on creating dialog boxes and linking them to the notebook using {DODIALOG}.

Program Flow

Use Program Flow commands to create macros that behave differently based on logical decisions, to break down complicated macros, or to consolidate repetitious macros into subroutines. The next figure shows a macro that deletes all negative values from the selected block.

Figure 3.4
A flow macro

The commands {FOR}, {IF}, {RETURN}, and {QUIT} in this example are program flow commands.

MACROS.WB1

	A	B	C	D	E	F	G	H
1	cell_value	0		cell_num	2			
2	row_num	4		num_block	[]A:C5..A:D8			
3								
4	del_negatives	{LET num_block,@PROPERTY("Active_Block.Selection")}					Store coordinates of active block	
5		{BlockName.Create temp, +"[]"&[MACROS]num_block}					Create a name from the active block	
6		{EditGoto temp}					Goto the block being checked	
7		{FOR row_num,0,@ROWS([]temp)-1,1,convert_row}					Check each row of the block	
8		{BlockName.Delete temp}					Delete the block name	
9		{QUIT}					Exit the macrc	
10								
11	convert_row	{FOR cell_num,0,@COLS([]temp)-1,1,check_cell}					Check each cell in the row	
12		{RETURN}					Try next row	
13								
14	check_cell	{LET cell_value,@CELLINDEX("contents",[]temp,cell_num,row_num)}					Store cell value	
15		{IF +cell_value<0}{BRANCH erase_cell}					Run erase_cell if the cell is negative	
16		{RETURN}					Return to subroutine convert_row	
17								
18	erase_cell	{BLANK @CELLINDEX("address",[]temp,cell_num,row_num)}					Erase the cell, since it is negative	
19		{RETURN}					Return to the subroutine check_cell	

Startup / datelab / ddeuse / delneg / entguide / indicell / looper / makepipe / commeq

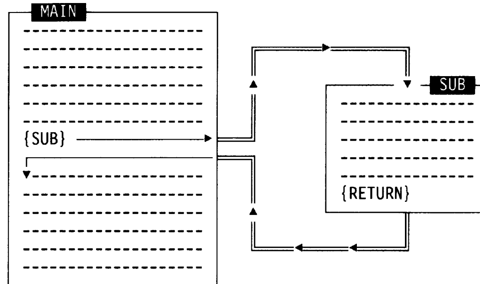
Subroutines

If you use many macros in a notebook you may find they often include the same sequence of macro commands. You can store these portions as *subroutines* to save time and memory. A subroutine is a macro stored separately from the macros using it. You name the subroutine just like other macros. You can *call* the subroutine from another macro, and Quattro Pro runs the subroutine's macro commands. After the subroutine runs, macro execution resumes in the main macro, running the command immediately following the command that called the subroutine. Using subroutines makes the macro more readable and easier to debug.

To call a subroutine from within a macro, type its name inside braces. For example, {go_right} calls the subroutine named go_right.

The following figure shows a main macro (MAIN) calling a subroutine (SUB). The {RETURN} command at the end of the subroutine returns control to MAIN. (Control also returns to a main macro when the subroutine encounters an empty cell, but using {RETURN} makes the macro clearer.) Then MAIN picks up where it left off by running the macro command immediately following the subroutine call.

Figure 3.5
Using a subroutine macro



Arguments and subroutines

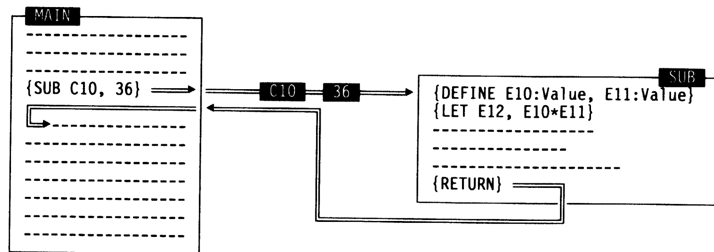
You can pass arguments to a subroutine for use by its commands. These arguments are stored in cells referenced by the subroutine. For example,

{set_cost C10,36}

calls the subroutine set_cost and passes two arguments (C10 and 36) to it. In order for the subroutine to know what to do with the arguments, you must define them within the subroutine using {DEFINE}, which tells Quattro Pro where to store the arguments and whether they should be interpreted as values or labels.

The following figure shows a macro passing two arguments (C10 and 36) to a subroutine. These arguments are immediately defined by a {DEFINE} command (in the first cell of the subroutine) as values stored in cells E10 and E11, respectively. The next line of the subroutine uses these values to calculate a third value, which is stored in E12.

Figure 3.6
Passing arguments to a subroutine



Whenever you call a subroutine, Quattro Pro stores the return point in an internal list called a *stack*. One return point on this stack clears when the subroutine encounters a {RETURN} command or empty cell. If you don't clear all of these locations, the stack fills up, causing the error `Too Many Nesting Calls.`

Use {BRANCH} instead of a subroutine when a macro

- doesn't need to pick up where it left off after calling the subroutine
- needs to return to a different point in the macro from the point right after the subroutine call
- calls the main macro as a subroutine

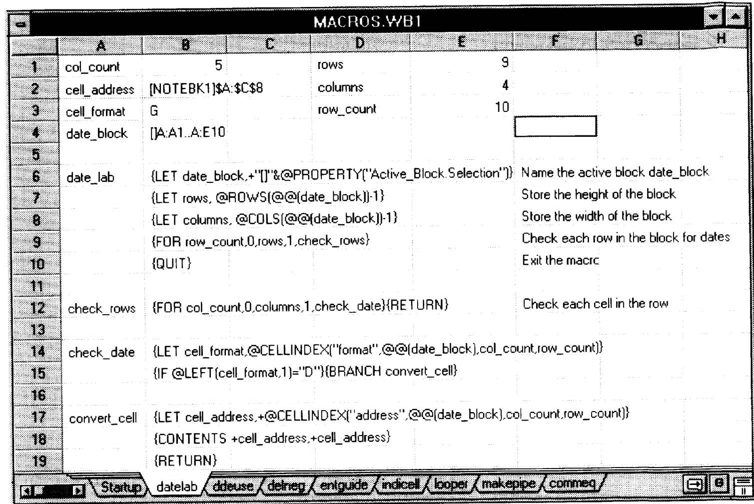
{BRANCH} can move to or return from any cell in a macro.

Cell

Cell commands can enter and read data, recalculate specific blocks, or convert values into formatted labels. The next figure shows a macro that traverses a block of dates and converts them into labels.

Figure 3.7
A cell macro at work

The commands {CONTENTS} and {LET} in this example are cell commands.



MACROS.WB1

	A	B	C	D	E	F	G	H
1	col_count	5		rows	9			
2	cell_address	[NOTEBK1]$A:$C$8		columns	4			
3	cell_format	G		row_count	10			
4	date_block	[]A:A1..A:E10						
5								
6	date_lab	{LET date_block,+"[]"&@PROPERTY("Active_Block.Selection")}				Name the active block date_block		
7		{LET rows, @ROWS(@@(date_block))-1}				Store the height of the block		
8		{LET columns, @COLS(@@(date_block))-1}				Store the width of the block		
9		{FOR row_count,0,rows,1,check_rows}				Check each row in the block for dates		
10		{QUIT}				Exit the macro		
11								
12	check_rows	{FOR col_count,0,columns,1,check_date}{RETURN}				Check each cell in the row		
13								
14	check_date	{LET cell_format,@CELLINDEX("format",@@(date_block),col_count,row_count)}						
15		{IF @LEFT(cell_format,1)="D"}{BRANCH convert_cell}						
16								
17	convert_cell	{LET cell_address,+@CELLINDEX("address",@@(date_block),col_count,row_count)}						
18		{CONTENTS +cell_address,+cell_address}						
19		{RETURN}						

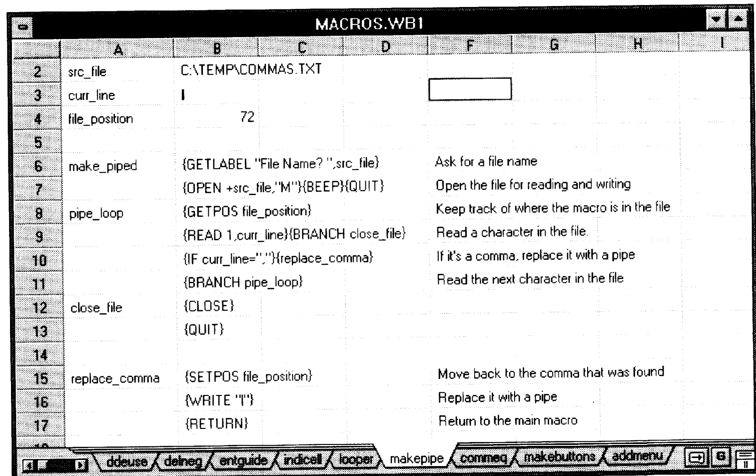
Startup / datelab / ddeuse / delneg / entguide / indicell / looper / makepipe / commeq

File

File commands can open external text files and read their data character by character, or create custom text files. The next figure shows a macro that opens a file and replaces all commas in the file with vertical bars (|).

Figure 3.8
A file macro

The commands {CLOSE}, {GETPOS}, {OPEN}, {READ}, {SETPOS} and {WRITE} in this example are file commands.



MACROS.WB1

	A	B	C	D	F	G	H	I
2	src_file	C:\TEMP\COMMAS.TXT						
3	curr_line	I						
4	file_position	72						
5								
6	make_piped	{GETLABEL "File Name? ",src_file}			Ask for a file name			
7		{OPEN +src_file,"M"}{BEEP}{QUIT}			Open the file for reading and writing			
8	pipe_loop	{GETPOS file_position}			Keep track of where the macro is in the file			
9		{READ 1,curr_line}{BRANCH close_file}			Read a character in the file			
10		{IF curr_line=","}{replace_comma}			If it's a comma, replace it with a pipe			
11		{BRANCH pipe_loop}			Read the next character in the file			
12	close_file	{CLOSE}						
13		{QUIT}						
14								
15	replace_comma	{SETPOS file_position}			Move back to the comma that was found			
16		{WRITE "I"}			Replace it with a pipe			
17		{RETURN}			Return to the main macro			

ddeuse / delneg / entguide / indicell / looper / makepipe / commeq / makebuttons / addmenu

If a file command succeeds, the macro command in the next *cell* is run, ignoring any other commands in the same cell as the file command. Commands in the same cell as the file command run

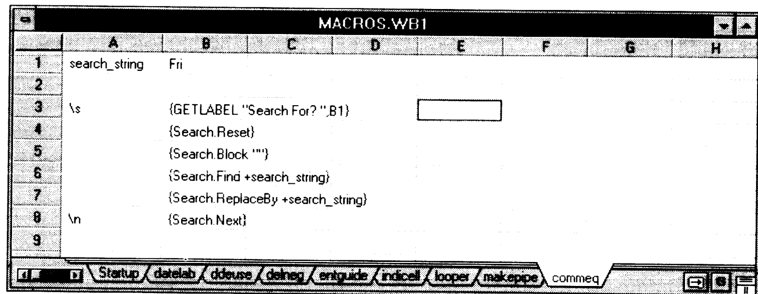
only if the file command fails. In the previous example, {BRANCH close_file} runs only if the read operation fails; {BEEP}{QUIT} runs only if a problem occurs while opening the file. Use this method to trap error conditions such as the disk being full or reading data past the end of the file.

Command Equivalents

The commands in this category perform operations normally done with menus or dialog boxes. Options normally set in a dialog box are passed as arguments to a command equivalent, like passing them to a subroutine. Command equivalents make macros easier to read and understand, and let the macro run in any Quattro Pro menu system.

Menu commands containing many settings (such as File | Print and Block | Fill) have a set of command equivalents that emulate their operation. The next figure shows a macro that emulates Edit | Search And Replace:

Figure 3.9
Command equivalents in action



Notice that each command equivalent sets one option in the Edit | Search And Replace dialog box. You don't have to set each option every time; if one of the command equivalents is omitted, the default setting is used. The final command ({Search.Next}) performs the fill operation. You can use command equivalent names with @COMMAND to find current Quattro Pro settings. For example, @COMMAND("BlockFill.Series") returns the current setting of Series in Block | Fill. Command equivalents that change Quattro Pro settings are *persistent* command equivalents. A persistent command equivalent typically contains a period in its name. The word after the period denotes an action to perform (like {Search.Next} in the previous example) or a setting to change (like the other command equivalents in the previous example).

Menu commands with a few settings (such as Block | Move and Block | Copy) have one command equivalent to emulate them. These are *nonpersistent* command equivalents. Command equivalents without a period are typically nonpersistent command equivalents. For example, the following macro command emulates Block | Copy:

{BlockCopy A1,A2..A36}

You can specify the selected block in a command equivalent; see the second example of @PROPERTY on page 82.

If you specify a command equivalent with a question mark (?) after the command name, the macro command displays a dialog box that the user can manipulate. If the name of the command equivalent contains a period, use only the part of the command name that precedes the period. For example, {BlockFill?} displays the Block | Fill dialog box for the user to manipulate. When the user chooses OK, the fill occurs and macro resumes running. Not all command equivalents support the question mark.

If you specify a command equivalent with an exclamation mark (!) after the command name, the macro command displays a dialog box that the *macro* can manipulate. You can make the dialog box revert to user control at any point by using the command {PAUSEMACRO}. Not all command equivalents support the exclamation mark.

Caution! Make sure that a dialog box is displaying or that Quattro Pro is in INPUT or FIND mode when the macro command runs {PAUSEMACRO}; otherwise the macro waits until it's stopped by *Ctrl+Break*.

The next figure shows a macro that uses {PAUSEMACRO} and {RestrictInput.Enter} to display a small form for the user to enter a name and phone number.

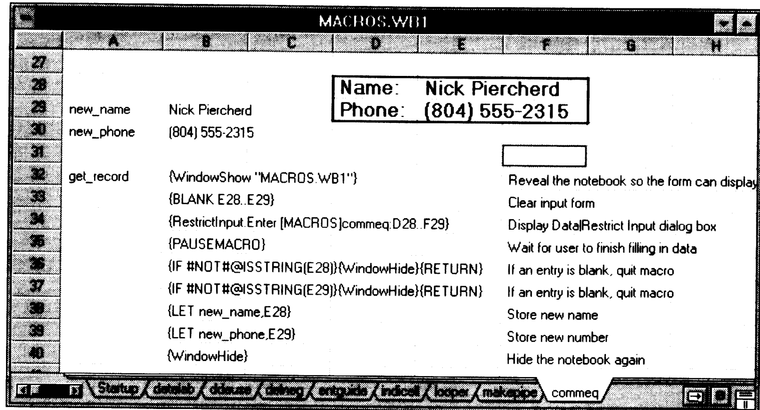


Figure 3.10
A form input macro

For a complete list of command equivalents, see Chapter 4 and Appendix A.

Dynamic Data Exchange (DDE)

Dynamic Data Exchange (DDE) lets Windows applications manipulate one another and communicate. With DDE, you open a channel of communication with another application using the {INITIATE} command. This is called initiating a *conversation*. Every DDE conversation consists of a *client* and a *server*. The application initiating the conversation is the client; the other application is the server. Once you've initiated a conversation, you can use {POKE} to send data, {REQUEST} to receive data, or {EXECUTE} to run macros in the server. {TERMINATE} ends the conversation. The {EXEC} command lets you run other Windows applications and execute DOS commands.

The next figure shows a macro that runs ObjectVision, initiates a conversation with TASKLIST.OVD (an ObjectVision file containing a list of tasks and their status), and stores each task and its status in the active notebook.

Figure 3.11
A macro to talk to ObjectVision

The commands {EXEC}, {INITIATE}, {REQUEST}, and {EXECUTE} in this example are DDE commands.

MACROS.WB1

	A	B	C	D	E	F	G	H
1	channel	1						
2	topics	0						
3	last_task	Call John RE: Quattro Pro						
4								
5	get_data	{EXEC "VISION C:\QPW\SAMPLES\TASKLIST.OVD",2}				Run ObjectVision and load TASKLIST.OVD		
6		{INITIATE "VISION","TASKLIST.OVD",channel}				Initiate a conversation with TASKLIST.OVD		
7		{EXECUTE channel,"[@BOTTOM(""tasks"")]",topics}				Go last entry in TASKLIST		
8		{REQUEST channel,"Task",last_task}				Store last task to check for end of table		
9		{EXECUTE channel,"[@TOP(""tasks"")]",topics}				Go to first entry in TASKLIST		
10	next_task	{REQUEST channel,"Task",[]C(0)R(0)}				Store task description in current cell		
11		{REQUEST channel,"Completed",[]C(1)R(0)}				Store task status next to current cell		
12		{IF +[]C(0)R(0)=last_task}{BRANCH end_session}				Check if this is the last entry		
13		{DOWN}				Move cell pointer down a row		
14		{EXECUTE channel,"[@NEXT(""tasks"")]",topics}				Make TASKLIST shift to next entry		
15		{BRANCH next_task}				Store next entry in current row		
16	end_session	{EXECUTE channel,"[@APPEXIT]",topics}				Close ObjectVision		
17								

Startup / datelab / ddeuse / delneg / entguide / indicell / looper / makepipe / commeq

The source of data that a DDE conversation connects with in the server application is called a *topic*. In the previous example, the file TASKLIST.OVD is a topic available from ObjectVision; Task and Completed are items of data available from that topic.

Quattro Pro as a DDE server

Many DDE applications support the topic System and items available from it.

Other DDE applications can call Quattro Pro as their server; to do so, use the server name `QPW` and specify `System` or the file name of an open notebook as the topic. For example, the following command initiates a conversation with the notebook OUTLINE.WB1, which is open on the Quattro Pro desktop:

```
{INITIATE "QPW","OUTLINE.WB1",A27}
```

When a notebook is the topic, you can enter cell addresses or block coordinates as the item to request. For example, the following macro command requests the contents of the block C3..C45.

```
{REQUEST Channel, "C3..C45", A3..A45}
```

Channel is the ID number of the DDE conversation.

When System is the topic, you can run Quattro Pro macros or request the items in the following table.

Table 3.2
Items available from the topic System

Item	Returns
SysItems	A list of the items you can request when System is the topic.
Topics	A list of topics currently available from the server application.
Status	The current status of the application. In Quattro Pro this is the text of the status line indicator (READY, WAIT, LABEL and so on). See Appendix A of the *User's Guide* for a description of each status indicator.
Formats	A list of Clipboard formats supported by the application through DDE.
Selection	The coordinates of the active block.

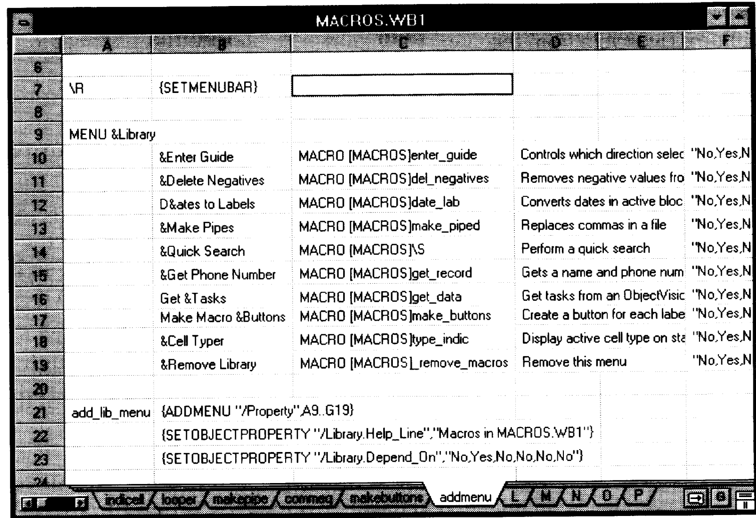
Other DDE applications can run macro commands in Quattro Pro. You can't run macros when Quattro Pro is busy performing a task, or in an area where macros normally won't run. For example, the following macro command displays a new notebook.

```
{EXECUTE Channel, "{FileNew}"}
```

UI Building

UI Building commands can add or remove menu commands from the active menu bar. For example, the macro shown in the following figure adds a menu to the menu bar. You can restore the standard menu bar using the macro \R:

Figure 3.12
Changing the menu bar



MACROS.WB1

	A	B	C	D	E	F
6						
7	\R	{SETMENUBAR}				
8						
9	MENU &Library					
10		&Enter Guide	MACRO [MACROS]enter_guide	Controls which direction selec		"No,Yes,N
11		&Delete Negatives	MACRO [MACROS]del_negatives	Removes negative values fro		"No,Yes,N
12		D&ates to Labels	MACRO [MACROS]date_lab	Converts dates in active bloc		"No,Yes,N
13		&Make Pipes	MACRO [MACROS]make_piped	Replaces commas in a file		"No,Yes,N
14		&Quick Search	MACRO [MACROS]\S	Perform a quick search		"No,Yes,N
15		&Get Phone Number	MACRO [MACROS]get_record	Gets a name and phone num		"No,Yes,N
16		Get &Tasks	MACRO [MACROS]get_data	Get tasks from an ObjectVisic		"No,Yes,N
17		Make Macro &Buttons	MACRO [MACROS]make_buttons	Create a button for each labe		"No,Yes,N
18		&Cell Typer	MACRO [MACROS]type_indic	Display active cell type on sta		"No,Yes,N
19		&Remove Library	MACRO [MACROS]_remove_macros	Remove this menu		"No,Yes,N
20						
21	add_lib_menu	{ADDMENU "/Property",A9..G19}				
22		{SETOBJECTPROPERTY "/Library.Help_Line","Macros in MACROS.WB1"}				
23		{SETOBJECTPROPERTY "/Library.Depend_On","No,Yes,No,No,No,No"}				

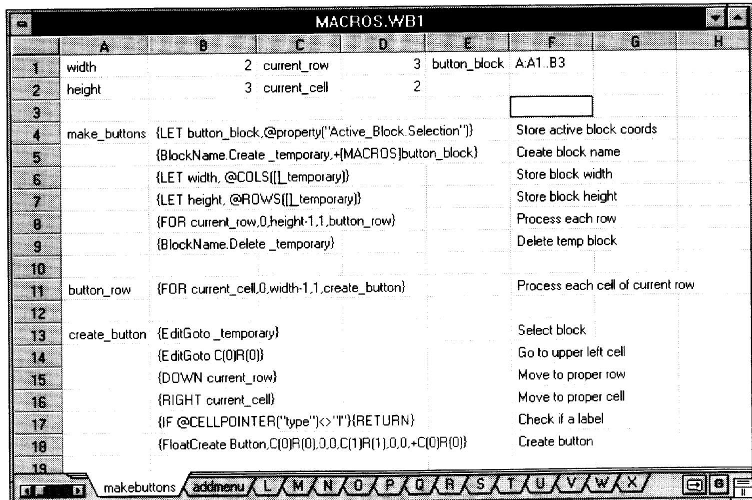
indicell / looper / makepipe / commsq / makebuttons / addmenu / L / M / N / O / P

See Chapter 7 for more information on UI Building commands.

Object

Object commands can create Quattro Pro objects, change their properties, or move them to new positions. The next figure shows a macro that creates macro buttons using the labels in the active block.

Figure 3.13
Creating macro buttons from labels



MACROS.WB1

	A	B	C	D	E	F
1	width	2	current_row	3	button_block	A:A1..B3
2	height	3	current_cell	2		
3						
4	make_buttons	{LET button_block,@property("Active_Block.Selection")}				Store active block coords
5		{BlockName.Create _temporary,+[MACROS]button_block}				Create block name
6		{LET width, @COLS([]_temporary)}				Store block width
7		{LET height, @ROWS([]_temporary)}				Store block height
8		{FOR current_row,0,height-1,1,button_row}				Process each row
9		{BlockName.Delete _temporary}				Delete temp block
10						
11	button_row	{FOR current_cell,0,width-1,1,create_button}				Process each cell of current row
12						
13	create_button	{EditGoto _temporary}				Select block
14		{EditGoto C(0)R(0)}				Go to upper left cell
15		{DOWN current_row}				Move to proper row
16		{RIGHT current_cell}				Move to proper cell
17		{IF @CELLPOINTER("type")<>"l"}{RETURN}				Check if a label
18		{FloatCreate Button,C(0)R(0),0,0,C(1)R(1),0,0,+C(0)R(0)}				Create button

makebuttons / addmenu / L / M / N / O / P / Q / R / S / T / U / V / W / X

Creating and positioning objects

After creating an object with a macro command, use {SETPROPERTY} to set its name or {GETPROPERTY} to store its name in a cell; then you can change its properties with macro commands at any time.

Object commands can create three types of objects:

Drawn objects are objects in a graph window that you'd normally create using the graph SpeedBar. When you're creating a drawn object with a macro command, the position for the object is specified by stating how many *pixels* it should appear from the upper left corner of the graph background. A pixel is the smallest dot that Windows can display on your screen. The following command activates a graph window and creates a line near the upper left corner of the graph:

```
{GraphEdit "PROFITS"}
{CREATEOBJECT "Line",0,0,25,25}
```

Dialog Controls are objects in a dialog window. When creating a dialog control using a macro command, the position is also specified by stating how many pixels it should appear from the upper left corner of the dialog window. For example, this command creates a push button in the active dialog window:

```
{CREATEOBJECT "Button",43,41,58,77}
```

Floating objects are macro buttons and floating graphs. Unlike graph objects and dialog controls, the position of a floating object is specified as an offset from a cell in the notebook. The offset is specified in *twips*; each twip is 1/1440th of an inch. For example, the following macro command creates a macro button that's a half an inch from left edge of A:A1:

```
{FLOATCREATE "Button",A:A1,720,0,A:B2,720,360,"Bt1"}
```

In this example, `A:A1,720,0` specifies that the upper left corner of the button is in A:A1, and is 720 twips from the left side of the cell (half an inch), as well as 0 twips from the top of the cell. `A:B2,720,360` specifies that the lower left corner of the button is in A:B2, 720 twips from the left side of A:B2, and a quarter inch (360 twips) from the top of A:B2.

Selecting, positioning, and sizing objects

After creating a dialog control or floating object, they're selected, so you can reposition them or change their property settings. There are three commands that select Quattro Pro objects:

You can use {SELECTBLOCK} to select a block and change or read its properties.

You can use {SELECTOBJECT} to select graph objects or dialog controls. When a graph object or dialog control is selected, you can use {MOVETO} and {RESIZE} to move and resize it. See the descriptions of these commands in Chapter 4 for details.

You can select a floating object with {SELECTFLOAT}. When a floating object is selected you can use {FLOATMOVE} and {FLOATSIZE} to move and resize it. See the descriptions of these commands in Chapter 4 for details.

Changing/reading property settings

You can use {SETPROPERTY} and {SETOBJECTPROPERTY} to change property settings of Quattro Pro objects. For example, the following macro selects a block and changes its text color.

```
{SELECTBLOCK A:A1..C22}
{SETPROPERTY "Text_Color",5}
```

You can also change a property setting without selecting the block using the following command:

```
{SETOBJECTPROPERTY "A:A1..C22.Text_Color",5}
```

You can use {GETPROPERTY} and {GETOBJECTPROPERTY} to read property settings. {GETPROPERTY} reads settings of the selected object; {GETOBJECTPROPERTY} lets you read property settings without selecting an object. See Appendix B for information on identifying objects and properties in a {GETOBJECTPROPERTY} command. Appendix B also contains a list of objects and properties you can manipulate with Object macro commands.

Accessing other notebooks

You can use linking in macros just as you would in formulas. Macros can run macros in other notebooks, access data from them, or store user input in them. For details on creating notebook links, see Chapter 12 of the *User's Guide*.

If you run a macro that's in a different notebook, such as a macro library, the macro behaves as though it were stored in the active notebook; block coordinates entered in dialog boxes refer to the active notebook. However, coordinates stored in the macro as arguments to a macro command refer to the macro library (except for command equivalents). For example, {BRANCH A1} branches to cell A1 of the macro library, not the active notebook. To refer to a block in a different notebook, use linking. For example, the

following command branches to page B, cell A1 of the notebook MACROS.

{BRANCH [MACROS]B:A1}

To refer to a block in the active notebook, precede its coordinates with empty brackets. For example, {BRANCH []A1} branches to cell A1 in the current page of the active notebook.

Note Block coordinates stored in a macro as arguments to a command equivalent *always* refer to the active notebook. To make them refer to coordinates in a macro library, use linking. This is also true for relative references (discussed next).

Relative references

Relative references are cell addresses specified as an offset from the cell selector.

You can also use relative references in formulas or dialog boxes.

For example, the relative reference of the cell selector is `[]P(0):C(0)R(0)`. The first piece of the reference, `[]`, specifies that the reference is relative to the cell selector. The second piece, `P(0):`, is the number of pages from the selector (in this case, zero); it's optional (except when selecting an entire row or column, which is discussed next). The third piece, `C(0)`, is the number of columns from the selector. The final piece, `R(0)`, is the number of rows from the selector. The cell below the selector is `[]C(0)R(1)`, the cell to the right of it is `[]C(1)R(0)`, and the cell beneath the selector (on the next notebook page) is in `[]P(-1):C(0)R(0)`.

You can omit C() or R() to select entire rows or columns. For example, {SELECTBLOCK []P(0):C(0)} selects the column containing the selector; {SELECTBLOCK []P(0):R(0)} selects the row containing the selector. To select the three columns to the right of the selector, use {SELECTBLOCK []P(0):C(1)..C(3)}.

Relative references can use negative or positive offsets. For example, if cell A2 is active, you could use `[]C(0)R(-1)..C(2)R(1)` to specify the block A1..C3. This addressing scheme increases macro portability, but hinders readability, since it's harder to track which blocks the macro affects. If your macros use @CELLPOINTER frequently, try relative references.

Caution! If [] doesn't precede a relative reference, the relative reference is offset from the cell containing the macro command, not the cell selector. For example, if {BLANK C(1)R(0)} was stored in A:A16, running it would erase the cell to the right of it (A:B16).

You can precede the relative reference with a colon (:) to affect the same cell, but on the active page. For example, if {BLANK :C(1)R(0)} was stored in A:A16, running it when page B is active erases the cell B:B16.

Self-modifying macros

You can use macro commands and text formulas to create *self-modifying macros* (also called *dynamic macros*) that alter themselves while running. For example, to save the active notebook under the name ACCT*x*.WB1, where *x* is a value stored in cell A5, enter the following text formula (discussed in Chapter 2 of the *User's Guide*) into a cell:

```
+"{FileSaveAs ""ACCT"&@STRING(A5,0)&".WB1""}"
```

The result of this formula runs as a macro command. For example, if cell A5 contains the value 5 in the previous example, Quattro Pro runs {FileSaveAs "ACCT5.WB1"}; if cell A5 changes to 10 and the notebook recalculates, Quattro Pro runs {FileSaveAs "ACCT10.WB1"}.

You can also use macro commands to change a portion of the macro while it's running. The next figure shows a macro that uses {GET} to convert user keystrokes into macro actions.

Figure 3.14
A macro that updates itself

MACROS.WB1

	A	B	G
1	cell_type	b	
2			
3	type_indic	{INDICATE @UPPER(@CELLPOINTER("type"))}	Set indicator to current cell type
4		{LET cell_type,@cellpointer("type")}	
5		{IF cell_type="l"}{INDICATE LABEL}	
6		{IF cell_type="b"}{INDICATE BLANK}	
7		{IF cell_type="v"}{INDICATE VALUE}	
8		{GET next_key}	Capture the next keystroke
9		{IF @UPPER(next_key)="{ESC}"}{BRANCH clear_indic}	Quit if it is Escape
10	next_key	{Right}	Execute the keystroke
11		{BRANCH addr_indic}	Update indicator, get next key
12			
13	clear_indic	{INDICATE}	Reset indicator back to its default
14		{QUIT}	Exit the macrc

indicell / looper / makepipe / commeq / makebuttons / addmenu / L / M / N / O / P

You can use these concepts to create powerful macros that completely change themselves based on their environment.

Debugging macros

Debugging is the process of isolating and fixing problems in a macro. Quattro Pro contains a powerful utility for debugging macros. With the *debugger*, you can

- run macros in slow motion (step by step), pausing as long as you want between steps
- set breakpoints to pause a macro when it reaches a given cell or satisfies a given condition
- run macros at full speed until reaching a breakpoint, then either continue in slow motion or at full speed until the next breakpoint
- monitor, or *trace*, changes to specific cells as a macro runs

Isolating a problem in a long macro isn't simple at regular speed. For this reason, the debugger uses a special mode called Debug. In Debug mode, Quattro Pro runs a macro step by step, pausing for a signal from the mouse or keyboard before going on to the next step. This way, you can monitor each phase of the macro.

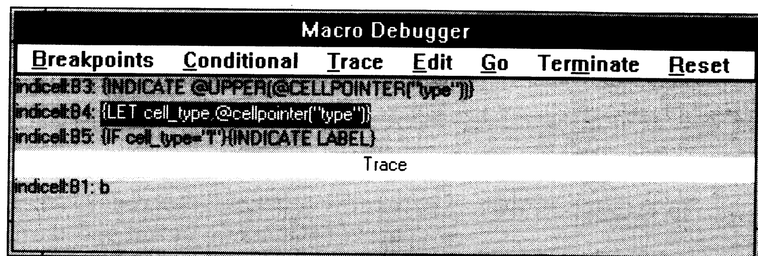
[Shift][F2] *Debug*

To run a macro in Debug mode, choose Tools | Macro | Debugger (or press *Pause* or *Shift+F2*). This activates Debug mode and displays the DEBUG indicator on the status line. Then run the macro to debug.

The debug window

When running a macro in Debug mode, a debug window appears in the bottom half of the screen (see the following figure). The first cell of the macro appears in the middle of the debug window, and the MACRO indicator appears on the status line. You can move or resize this window.

Figure 3.15
The macro debug window



The debug window is divided into two sections:

- **The top (macro) section** contains at most three rows. The middle row displays the macro cell running. The top row displays the previous macro cell, if any, and the third row displays the next macro cell, if any. The macro command about to run is highlighted.
- **The bottom (trace) section** displays the contents of trace cells specified using Trace, and it shows how the macro affects them. Trace cells are discussed on page 143.

The debugger menu

The menu bar at the top of the debug window shows the following debugger commands:

- **Breakpoints** lets you specify up to four cells or blocks where you want to pause the macro.
- **Conditional** lets you specify up to four cells that contain logical formulas, such as +A3=10; when the formula becomes true, the macro returns to running step-by-step.
- **Trace** lets you specify up to four cells whose contents you want to monitor while a macro is being debugged.
- **Edit** lets you change the macro you're debugging without leaving DEBUG mode.
- **Go** runs the macro until it reaches a breakpoint or finishes.
- **Terminate** stops the macro and removes the debug window from the screen (this is the same as pressing *Ctrl+Break*).
- **Reset** removes breakpoints and trace cells set with the Breakpoints, Conditional, and Trace commands.

Stepping through a macro

To run the first command of the macro, click in the debug window or press *Spacebar*. Clicking repeatedly steps through each macro command until an error is pinpointed. To run the rest of the macro at full speed, choose Go or press *Enter*.

Setting breakpoints

If you know most of a macro is correct, you can use *breakpoints* to run parts of the macro at full speed and then enter Debug mode. *Standard breakpoints* pause the macro when the breakpoint cell is

reached. *Conditional breakpoints* pause the macro when the result of a logical formula in the conditional breakpoint cell is true. Up to four standard breakpoints and four conditional breakpoints can be set at one time.

Standard breakpoints

When debugging a macro containing a standard breakpoint, Quattro Pro runs the macro at full speed until it reaches a breakpoint, then pauses the macro. To resume stepping through the macro, click in the debug window; to continue at full speed until the next breakpoint or the end of the macro, choose Go.

Note If there are several macro commands in the breakpoint cell, you may need to choose Go more than once to make the macro run at full speed.

To set a standard breakpoint,

1. Choose Breakpoints. Quattro Pro displays four breakpoints.
2. Choose the breakpoint to set.
3. Choose Location and specify the cell or block where you want the macro to stop. If you specify a block, Quattro Pro uses its first cell.
4. Sometimes a problem appears after many repetitions of the same commands in a loop macro. If this is the case, you can set a pass count to indicate how many times to pass through the breakpoint before stopping. Choose Pass Count and specify the number of passes. The default, 1, tells Quattro Pro to stop every time it passes through the breakpoint. Setting it to 2 makes Quattro Pro stop every other pass. Choose OK to set the breakpoint and return to the debug window.
5. Repeat steps 1 through 4 to set additional breakpoints.

You can disable a standard breakpoint by setting its pass count to zero.

Breakpoints in action

The macro in the following figure includes a loop that continuously increments a single-cell block called COUNTER.

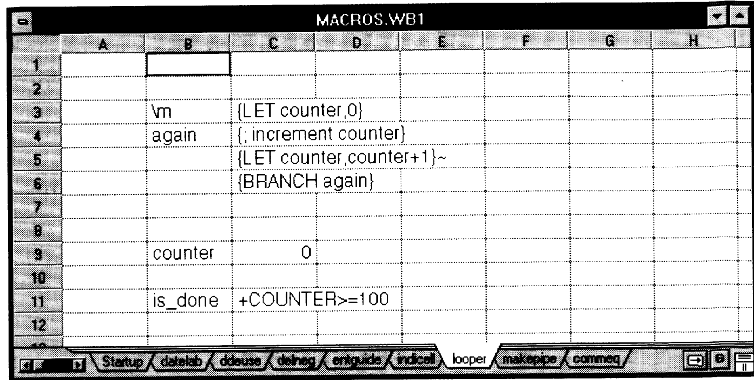


Figure 3.16
An example macro loop

If you specify C6 as the first breakpoint and leave the pass count at 1, the macro stops at the {BRANCH} command each time it goes through the loop. When you choose Go, it continues, incrementing the counter cell by one. In the top of the debug window, the selector highlights the {BRANCH} command, indicating that it's the next command.

If you specified a pass count of 5 for the first breakpoint in this example, then every time you choose Go, five loops occur, and the counter increments from 0 to 5 to 10 and so on.

Conditional breakpoints

Conditional breakpoints stop the macro when the result of a logical formula becomes true. In the previous figure, cell C11 contains a logical formula. (The cell containing the formula is formatted so that the formula, rather than the value calculated, appears.) C11 is false (has a value of 0) until 100 or more loops occur. If this cell is specified as a conditional cell, the macro pauses when the counter reaches 100.

You can set up to four conditional breakpoints per notebook. To set a conditional breakpoint,

1. Choose Conditional to display a menu of conditional breakpoints.
2. Choose the conditional breakpoint to set.
3. Specify the cell containing the condition, and choose OK to set the breakpoint and return to the debug window.
4. Repeat steps 1 through 3 to set additional conditional breakpoints.

Conditional breakpoints can be extremely valuable. Suppose you have a macro loop that does some date calculations and you want it to pause whenever December is reached. If the date is in cell A5, assign the first conditional breakpoint to a cell containing the following formula:

```
@MONTH(A5)=12
```

As long as the formula in the condition cell is true (has a value of 1), the macro runs in step-by-step mode. Once the formula is false (has a value of 0), choose Go to let the macro continue until the next conditional formula evaluates as true. This process repeats until all conditional breakpoints are passed.

Monitoring cells

Macros often affect the contents of one or more cells. By *tracing* these cells, you can see what the macro is doing. Quattro Pro lets you specify up to four *trace cells*, whose addresses and current values show during debugging in the debug window. Quattro Pro updates the debug window as the contents of the trace cells change.

To set a trace cell,

1. Choose Trace from the debug window.
2. Choose the trace cell to set (1st Cell through 4th Cell).
3. Specify the cell to trace and choose OK to set the trace cell and return to the debug window.

To specify additional trace cells, repeat these steps.

In the example in Figure 3.16, if the counter cell (C9) was specified as a trace cell, you could watch the counter increment during each loop.

Editing macro cells

You can use Edit on the debug window to edit the contents of a cell without leaving Debug mode:

1. Choose Edit from the debug window.
2. Choose Cell and point to or enter the address of the cell you want to edit.

3. Choose Contents. This edit field contains the contents of the cell you previously specified with Cell.
4. Edit the text that appears in Contents.
5. Choose OK to write the change back into the cell and return to the debug window.

Clearing trace cells and breakpoints

To remove all breakpoints (standard and conditional) and trace cells set for the notebook, choose Reset from the debug window.

Exiting Debug mode

When a macro finishes running in Debug mode, the debug window closes, but Debug mode remains in effect, as shown by the DEBUG indicator. To exit Debug mode, choose Tools | Macro | Disable Debugger, press *Shift+F2*, or press *Pause*.

To stop a macro before debugging finishes, choose Abort from the debug window or press *Ctrl+Break*. Then you can debug another macro or exit Debug mode.

Note When the debug window displays, part of the notebook is obscured. You can move the debug window to see the data it covers.

C H A P T E R

4

Macro command reference

Macro commands by type

Macro commands fall into the following categories. See the page in parentheses for a list of the commands in each category.

- **Keyboard** commands emulate the action of various keys on the keyboard (page 146).
- **Screen** commands affect the display (page 147).
- **Interactive** commands let you create macros that pause for the user to enter data from the keyboard (page 148).
- **Program flow** commands let you branch and loop in a macro (page 148).
- **Cell** commands affect the data stored in specified cells (page 149).
- **File** commands work with data within files other than the active notebook file (page 150).
- **DDE** commands manipulate other Windows applications (page 150).
- **UI building** commands let you alter the menu system (page 150).
- **Object** commands let you create, move, resize, select, and manipulate objects (page 151).

- **Miscellaneous** commands insert braces, tildes, comments, and blank lines in macros and scroll the window a specified distance (page 151).
- **Command equivalents** emulate operations you can choose from the Quattro Pro for Windows menu (page 152 and Appendix A).
- **/ Commands** emulate operations you can choose from the Quattro Pro for DOS menu. For a list of these menu equivalents, see the Quattro Pro for DOS *@Functions and Macros* manual.

Click the Macros button on the input line SpeedBar or press *Shift+F3*, the Macros key, for a menu of these macro command categories.

The following tables list the commands in each category, along with the syntax and a brief description. Italicized words describe the type of information expected; arguments in angle brackets (<>) are optional.

The next section, "Macro command descriptions," lists and describes each command.

Table 4.1: Keyboard commands listed by category

Keyboard command	Description
Status keys	
{CAPOFF}	*Caps Lock* off
{CAPON}	*Caps Lock* on
{INS} or {INSERT}	Toggles *Ins* on or off
{INSOFF}	*Ins* off
{INSON}	*Ins* on
{NUMOFF}	*Num Lock* off
{NUMON}	*Num Lock* on
{SCROLLOFF}	*Scroll Lock* off
{SCROLLON}	*Scroll Lock* on
Cursor-movement keys	
{HOME}	*Home*
{END}	*End*
{LEFT *<Number>*} or {L *<Number>*}	←
{RIGHT *<Number>*} or {R *<Number>*}	→
{UP *<Number>*} or {U *<Number>*}	↑
{DOWN *<Number>*} or {D *<Number>*}	↓
{PGUP *<Number>*}	*PgUp*
{PGDN *<Number>*}	*PgDn*
{BIGLEFT *<Number>*} or {BACKTAB *<Number>*}	*Ctrl* ← or *Shift+Tab*
{BIGRIGHT *<Number>*} or {TAB *<Number>*}	*Ctrl* → or *Tab*

Table 4.1: Keyboard commands listed by category (continued)

Keyboard command	Description
Function keys	
{ABS}	*F4*
{CALC}	*F9*
{EDIT}	*F2*
{FUNCTIONS}	*Alt+F3*
{GRAPH}	*F11*
{GRAPHPAGEGOTO}	*Shift+F5*
{HELP}	*F1*
{INSPECT}	*F12*
{MACROS}	*Shift+F3*
{MARK}	*Shift+F7*
{MENU}	*F10*
{NAME}	*F3*
{NEXTPANE}	*F6*
{NEXTTOPWIN}	*Ctrl+F6*
{NEXTWIN}	*Shift+F6*
{QGOTO}	*F5*
{QUERY}	*F7*
{STEPON}, {STEPOFF}	*Shift+F2*
{TABLE}	*F8*
Other keys	
{ALT+*Key* <*Number*>}	The keystroke *Key* while *Alt* is held down
{BACKSPACE <*Number*>} or {BS <*Number*>}	*Backspace*
{BREAK}	Returns Quattro Pro to Ready mode
{CLEAR}	*Ctrl+Backspace* (Clears any existing entry from a prompt line or from the input line in Edit mode)
{CR} or ~	*Enter*
{CTRL+*Key* <*Number*>}	The keystroke *Key* while *Ctrl* is held down
{DATE}	*Ctrl+Shift+D*
{DEL} or {DELETE}	*Del*
{ESC} or {ESCAPE}	*Esc*
{SHIFT+*Key* <*Number*>}	The keystroke *Key* while *Shift* is held down

For information about what a particular key does, see Appendix A in the *User's Guide*.

Table 4.2: Screen commands

Screen commands	Description
{BEEP <*Number*>}	Sounds the computer's bell
{INDICATE *String*}	Sets the mode indicator to display *String*
{PANELOFF}	Suppresses menu and prompt display
{PANELON}	Restores menus and prompts disabled by {PANELOFF}
{WINDOWSOFF}	Prevents screen updating
{WINDOWSON}	Restores screen updating disabled by {WINDOWSOFF}
{ZOOM}	Maximizes or restores the active window

Table 4.3: Interactive commands

Interactive commands	Description
{?}	Pauses the macro and accepts input from the keyboard until you press *Enter*
{ACTIVATE *WindowName*}	Makes the specified window active
{BREAKOFF}	Disables the *Ctrl+Break* key used to cancel a macro
{BREAKON}	Restores the *Ctrl+Break* key disabled by {BREAKOFF}
{CHOOSE}	Displays a pick list of open windows
{DODIALOG *Dialog,OKExit?, <Arguments>,<MacUse?>*}	Displays the dialog box *Dialog* for use
{GET *Location*}	Pauses the macro, accepts one keystroke and stores it in *Location*
{GETLABEL *Prompt,Location*}	Pauses the macro, displays *Prompt*, and stores the subsequent keystrokes as a label in *Location*
{GETNUMBER *Prompt,Location*}	Pauses the macro, displays *Prompt*, and stores the subsequent keystrokes as a numeric value in *Location*
{GOTO}	Moves the selector to a specified location
{GRAPHCHAR *Location*}	Stores the character a user presses to leave a graph or message in *Location*
{IFKEY *String*}	Returns true if *String* is the macro name for any key macro command
{LOOK *Location*}	Places the first keystroke typed in *Location*
{MENUBRANCH *Location*}	Passes macro control to a custom menu at *Location*
{MENUCALL *Location*}	Pauses the macro, and runs a customized menu as a subroutine
{MESSAGE *Block,Left,Top,Time*}	Displays contents of block in a pop-up window for *Time* seconds
{PAUSEMACRO}	Pauses the macro for dialog box entries, then resumes the macro when the user closes the dialog box
{STEPOFF}	Exits Debug mode, running the macro at normal pace
{STEPON}	Enters Debug mode, in which Quattro Pro runs the macro step by step, advancing when you press the spacebar
{UNDO}	"Takes back" the last command; cancels its effects
{WAIT *DateTimeNumber*}	Pauses the macro until the time specified in *DateTimeNumber* arrives
{WINDOW},{WINDOW*<Number>*}	Makes the next pane active; makes the specified window active

Table 4.4: Program flow commands

Program flow commands	Description
{BRANCH *Location*}	Passes execution control to another macro at *Location*
{DEFINE *Location1<:Type1>,...*}	Defines the type of arguments passed to a subroutine and tells where to store them
{DISPATCH *Location*}	Branches indirectly to an alternate macro branch
{FOR *CounterLoc,Start#,Stop#,Step#,StartLoc*}	Runs a subroutine the specified number of times
{FORBREAK}	Terminates execution of a {FOR} command

Table 4.4: Program flow commands (continued)

Program flow commands	Description
{IF *Condition*}	Checks to see if a condition is TRUE or FALSE before continuing execution
{ONERROR *BranchLocation,<MessageLocation>, <ErrorLocation>*}	Continues execution at a specified location after Quattro Pro detects an error
{QUIT}	Terminates macro execution and returns keyboard control
{RESTART}	Changes the current subroutine to the starting or main routine
{RETURN}	Terminates the current subroutine and returns control to main routine
{*Subroutine <ArgumentList>*}	Calls the specified subroutine, and passes any given arguments

Table 4.5: Cell commands

<table>
<tr><th>Cell commands</th><th>Description</th></tr>
<tr><td>{BLANK Location}</td><td>Erases a cell or block</td></tr>
<tr><td>{CONTENTS Dest,Source,<Width#>,<Format#>}</td><td>Copies the contents of one cell to another as a formatted label</td></tr>
<tr><td>{GETDIRECTORYCONTENTS Block, <Path>}</td><td>Enters into Block an alphabetized list of files in the directory specified by Path; if Path is not included, lists the current directory</td></tr>
<tr><td>{GETWINDOWLIST Block}</td><td>Lists open windows in Block</td></tr>
<tr><td>{LET Location,Value<:Type>}</td><td>Places a number or string in the given location; Type lets you store Value as a number or string</td></tr>
<tr><td>{PUT Location,Column#,Row#,Value<:Type>}</td><td>Places a number or string in a given location offset by the specified number of columns and rows; Type lets you store Value as a number or string</td></tr>
<tr><td>{PUTBLOCK Value,<Block>}</td><td>Stores Value in each cell of Block</td></tr>
<tr><td>{PUTCELL Data}</td><td>Stores Data in the active cell</td></tr>
<tr><td>{RECALC Location,<Condition>,<Iteration#>}</td><td>Recalculates formulas in Location in rowwise order for a specified number of times, or until Condition is reached</td></tr>
<tr><td>{RECALCCOL Location,<Condition>,<Iteration#>}</td><td>Recalculates formulas in Location in columnwise order for a specified number of times, or until Condition is reached</td></tr>
<tr><td>{SPEEDFILL}</td><td>Fills the selected block with sequential data</td></tr>
<tr><td>{SPEEDFORMAT FmtName, NumFmt? (0 | 1), Font? (0 | 1), Shading? (0 | 1), TextColor? (0 | 1), Align? (0 | 1), LineDraw? (0 | 1), AutoWidth? (0 | 1), ColHead? (0 | 1), ColTotal? (0 | 1), RowHead? (0 | 1), RowTotal? (0 | 1)}</td><td>Applies a standard format to the selected block</td></tr>
<tr><td>{SPEEDSUM Block}</td><td>Sums filled cells in Block, writes results in adjacent empty cells (right/bottom)</td></tr>
</table>

Table 4.6: UI Building commands

UI Building commands	Description
{ADDMENU *MenuPath,MenuBlock*}	Adds the menu *MenuBlock* to the menu system at *MenuPath*
{ADDMENUITEM *MenuPath,Name, <Link>,<Hint>,<HotKey>, <DependString>,<Checked* (Yes | No)>}	Adds a menu command to the menu system at *MenuPath*
{DELETEMENU *MenuPath*}	Deletes the menu *MenuPath* from the active menu system
{DELETEMENUITEM *MenuPath*}	Deletes the menu item *MenuPath* from the active menu system
{SETMENUBAR *SystemDefinition*}	Sets the active menu system

Table 4.7: File commands

File commands	Description
{ANSIREAD *#Bytes,Location*}	Reads the specified number of bytes and stores them in *Location*, without any character mapping
{ANSIREADLN *Location*}	Reads a line of characters and stores it in *Location*, without any character mapping
{ANSIWRITE *String,<String2,...>*}	Writes a string of characters to the current file, without any character mapping
{ANSIWRITELN *String,<String2,...>*}	Writes a string of characters to the current file and ends it with a carriage return/linefeed, without any character mapping
{CLOSE}	Closes a file opened by {OPEN}
{FILESIZE *Location*}	Calculates the number of bytes in the current file, and stores the value in *Location*
{GETPOS *Location*}	Places the position of the file pointer in *Location*
{OPEN *Filename,AccessMode*}	Opens a file for reading, writing, modifying, or appending
{READ *#Bytes,Location*}	Reads the specified number of bytes and stores them in *Location*
{READLN *Location*}	Reads a line of characters and stores it in *Location*
{SETPOS *FilePosition*}	Sets the file pointer to the value of *FilePosition*
{WRITE *String,<String2,...>*}	Writes a string of characters to the current file
{WRITELN *String,<String2,...>*}	Writes a string of characters to the current file and ends it with a carriage return/linefeed

Table 4.8: DDE commands

DDE commands	Description
{EXEC *AppName, WindowMode,<ResultLoc>*}	Runs the application *AppName*
{EXECUTE *DDEChannel, Macro,<ResultLoc>*}	Runs the macro *Macro* in a server application
{INITIATE *AppName, Topic,ChannelLoc*}	Initiates a conversation with a DDE application
{POKE *DDEChannel, Destination, DataToSend*}	Sends data to a DDE application
{REQUEST *DDEChannel, DataToReceive,DestBlock*}	Requests data from a DDE application
{TERMINATE *DDEChannel*}	Ends a DDE conversation

Table 4.9: Object commands

Object commands	Description
{COLUMNWIDTH *Block, FirstPane?* (0 | 1), *Set/Reset/Auto* (0 | 1 | 2), *Size*}	Sets the width of columns in *Block*
{CREATEOBJECT *ObjectName,x1,y1,x2,y2,* <*x3, y3,...*>}	Creates graph or dialog objects
{FLOATCREATE *Type,UpperCell,xoffset,yoffset, LowerCell,xoffset2,yoffset2,Text*}	Creates a floating graph or macro button
{FLOATMOVE *UpperCell,xoffset,yoffset*}	Moves a floating object in the active notebook window
{FLOATSIZE *UpperCell,xoffset,yoffset,LowerCell, xoffset2,yoffset2*}	Resizes a selected floating object in the active notebook window
{GETOBJECTPROPERTY *Cell,Object.Property*}	Gets property setting from an object
{GETPROPERTY *Cell,Property*}	Gets property setting from the object currently selected
{MOVETO *x,y*}	Moves selected objects to coordinates *x, y*
{RESIZE *x,y,NewWidth,NewHeight,*<*VertFlip?*>, <*HorizFlip?*>}	Resizes selected objects in the active window
{ROWCOLSHOW *Block, Show?* (0 | 1), *Row* (1) *or Col* (0), *FirstPane?* (0 | 1)}	Shows or hides rows or columns in *Block*
{ROWHEIGHT *Block, FirstPane?* (0 | 1), *Set/Reset* (0 | 1), *Size*}	Sets the height of rows in *Block*
{SELECTBLOCK *Block*}	Selects a contiguous or noncontiguous block within the active notebook
{SELECTFLOAT *ObjectID1*<*,ObjectID2, ...*>}	Selects floating objects in the active notebook window using their object names
{SELECTOBJECT *ObjectID1*<*,ObjectID2, ...*>}	Selects objects in active window
{SETGRAPHATTR *FillColor, BkgColor, FillStyle, BorderColor, BoxType*}	Equivalent to setting Graph Setup and Background properties (colors are comma-separated RGB triplets in quotes)
{SETOBJECTPROPERTY *Object.Property,Value*}	Sets properties of objects
{SETPROPERTY *Property,Setting*}	Sets properties of selected object

Table 4.10: Miscellaneous commands

Miscellaneous commands	Description
} or { } }	Inserts a right brace, }
{ { }	Inserts a left brace, {
{ ~ }	Inserts a tilde, ~
{ }	Does nothing; lets you enter a blank line in a macro without stopping macro execution
{ ; *String*}	Lets you enter explanatory remarks in a macro
{HLINE *Distance*}	Scrolls window horizontally *Distance* lines
{HPAGE *Distance*}	Scrolls window horizontally *Distance* pages
{VLINE *Distance*}	Scrolls window vertically *Distance* lines
{VPAGE *Distance*}	Scrolls window vertically *Distance* pages

For slide-show control commands, see Slide.*Option* in the following command equivalent list.

Table 4.11: Command equivalents

Command equivalent	Command
{Align.*Option*}	Dialog | Align | *Option*
{Application.*Property*}	Property | Application
{BlockCopy *Option(s)*}	Block | Copy
{BlockDelete.*Option*}	Block | Delete | *Options*
{BlockFill.*Option*}	Block | Fill | *Options*
{BlockInsert.*Option*}	Block | Insert | *Options*
{BlockMove *Option(s)*}	Block | Move
{BlockMovePages *Option(s)*}	Block | Move Pages
{BlockName.*Option*}	Block | Names | *Options*
{BlockReformat *Option(s)*}	Block | Reformat
{BlockTranspose *Option(s)*}	Block | Transpose
{BlockValues *Option(s)*}	Block | Values
{ClearContents *Option(s)*}	Edit | Clear Contents
{Controls.*Option*}	Dialog | Order | *Options*
{DialogView *Window*}	Tools | UI Builder
{DialogWindow.*Property*}	Property | Dialog
{EditClear}	Edit | Clear
{EditCopy}	Edit | Copy
{EditCut}	Edit | Cut
{EditGoto *Option(s)*}	Edit | Goto
{EditPaste}	Edit | Paste
{ExportGraphic *Option(s)*}	Draw | Export
{FileClose *Option(s)*}	File | Close
{FileCloseAll *Option(s)*}	File | Close All
{FileCombine *Option(s)*}	Tools | Combine
{FileExit *Option(s)*}	File | Exit
{FileExtract *Option(s)*}	Tools | Extract
{FileImport *Option(s)*}	Tools | Import
{FileNew}	File | New
{FileOpen *Option(s)*}	File | Open
{FileRetrieve *Option(s)*}	File | Retrieve
{FileSave *Option(s)*}	File | Save
{FileSaveAll *Option(s)*}	File | Save All
{FileSaveAs *Option(s)*}	File | Save As
{FloatOrder.*Option*}	Block | Object Order | *Options*
{Frequency.*Option*}	Data | Frequency | *Options*
{GraphCopy *Option(s)*}	Graph | Copy
{GraphDelete *Option(s)*}	Graph | Delete
{GraphEdit *Option(s)*}	Graph | Edit
{GraphNew *Option(s)*}	Graph | New
{GraphSettings.Titles *Option(s)*}	Graph | Titles
{GraphSettings.Type *Option(s)*}	Graph | Type
{GraphView *<Option(s)>*}	Graph | View
{GraphWindow.*Property*}	Property | Graph Window
{Group.*Option*}	Tools | Define Group | *Options*

Table 4.11: Command equivalents (continued)

Command equivalent	Command
{GroupObjects}	Draw | Group
{ImportGraphic *Option(s)*}	Draw | Import
{InsertBreak}	Block | Insert Break
{InsertObject *Option(s)*}	Edit | Insert Object
{Invert.*Option*}	Tools | Advanced Math | Invert | *Options*
{Links.*Option*}	Tools | Update Links | *Options*
{Multiply.*Option*}	Tools | Advanced Math | Multiply | *Options*
{NamedStyle.*Option*}	Edit | Define Style | *Options*
{NoteBook.*Property*}	Property | Active Notebook
{OLE.*Option*}	OLE Object Inspectors
{Optimizer.*Option*}	Tools | Optimizer | *Options*
{Order.*Option*}	Dialog | Order | *Options* and Draw | *OrderOptions*
{Page.*Property*}	Property | Active Page
{Parse.*Option*}	Data | Parse | *Options*
{PasteFormat *Option(s)*}	Edit | Paste Format
{PasteLink}	Edit | Paste Link
{PasteSpecial *Option(s)*}	Edit | Paste Special
{Preview}	File | Print Preview
{Print.*Option*}	File | Print | *PrintOptions*, File | Page Setup | *SetupOptions*, and File | Named Settings | *Options*
{PrinterSetup *Option(s)*}	File | Printer Setup
{Query.*Option*}	Data | Query | *Options*
{Regression.*Option*}	Tools | Advanced Math | Regression | *Options*
{ResizeToSame}	Dialog | Align | Resize to Same
{RestrictInput.Enter}	Data | Restrict Input
{RestrictInput.Exit}	any operation that ends INPUT mode
{Search.*Option*}	Edit | Search and Replace | *Options*
{Series.*Option*}	Graph | Series | *Options*
{Slide.Effect *Effect*}	Specifies the transition effect to use when displaying a slide in a slide show; no equivalent command
{Slide.Goto *SlideName*}	Takes the active slide show directly to the slide *SlideName*; no equivalent command
{Slide.Next}	Advances the active slide show to the next slide; no equivalent command
{Slide.Previous}	Returns the active slide show to the previous slide; no equivalent command
{Slide.Run *SlideShowName*}	Graph | Slide Show
{Slide.Speed *Speed*}	Specifies the transition speed to use when displaying a slide in a slide show; no equivalent command
{Slide.Time *Time*}	Specifies the time in seconds to display a slide in a slide show; no equivalent command
{SolveFor.*Option*}	Tools | Solve For | *Options*
{Sort.*Option*}	Data | Sort | *Options*
{TableQuery.Destination *Block*}	Data | Table Query | Destination
{TableQuery.FileQuery Yes | No}	Data | Table Query | Query in File

Table 4.11: Command equivalents (continued)

Command equivalent	Command
{TableQuery.Go}	Data | Table Query | OK
{TableQuery.QueryBlock *Block*}	Data | Table Query | Query in Block (checked)
{TableQuery.QueryFile *Filename*}	Data | Table Query | QBE File
{TableView}	Data | Database Desktop
{UngroupObjects}	Draw | UnGroup
{WhatIf.*Option*}	Data | What-If | *Options*
{WindowArrIcon}	Window | Arrange Icons
{WindowCascade}	Window | Cascade
{WindowClose}	Control Menu | Close
{WindowHide}	Window | Hide
{WindowMaximize}	Control Menu | Maximize
{WindowMinimize}	Control Menu | Minimize
{WindowMove *Option(s)*}	Control Menu | Move
{WindowNewView}	Window | New View
{WindowNext}	Control Menu | Next
{WindowPanes *Option(s)*}	Window | Panes
{WindowRestore}	Control Menu | Restore
{WindowShow *Option(s)*}	Window | Show
{WindowSize *Option(s)*}	Control Menu | Size
{WindowTile}	Window | Tile
{WindowTitles *Option(s)*}	Window | Locked Titles
{Workspace.*Option*}	File | Workspace | *Options*

See Appendix A for a list of command equivalents organized by menu. You can use many of the command equivalent names with @COMMAND to find current Quattro Pro settings.

For example, the following formula returns the current setting of Find in the Edit | Search and Replace dialog box:

@COMMAND("Search.Find")

See Chapter 2 for a description of @COMMAND.

The /x commands

Quattro Pro accepts the /x commands used by other spreadsheet programs. These commands all have macro command equivalents and work the same as those equivalents:

Table 4.12
/x commands and the corresponding macro commands

/x command	Macro command equivalent
/xc	{Subroutine}
/xg	{BRANCH}
/xi	{IF}
/xl	{GETLABEL}
/xm	{MENUBRANCH}
/xn	{GETNUMBER}
/xq	{QUIT}
/xr	{RETURN}

The arguments for /x commands are entered directly after the command. For example,

```
/xlName?~C10~
```

displays a prompt that reads, "Name?", then stores the response in cell C10. The tilde (~) commands insert carriage returns after the prompt and the cell address. Tildes are required after each argument.

Unlike their macro command equivalents, /x commands don't require a location parameter. If no location is included, the current cell is assumed, but the tilde is still required. For example,

```
/xlName?~~
```

Quattro Pro includes /x commands for macro compatibility with other spreadsheet products. Because of their lack of readability and nonstandard syntax, they aren't recommended.

Macro command descriptions

The rest of this chapter describes each macro command. Each entry shows the format to use. Items in italics are arguments that are specified when entering the command; the italicized words describe what information is expected. If an argument ends with a question mark (?), a yes or no answer is expected; enter 1 for yes

or 0 for no. Arguments enclosed in angle brackets (<>) are optional.

Quattro Pro records a variety of operations as special macro commands called *command equivalents*. They are summarized in the table on page 152 and are listed by menu command in Appendix A. Page 129 contains additional information on command equivalents and their syntax.

Some macro commands are equivalent to function keys and other special keys. If you need information about what a particular key does, refer to Appendix A in the *User's Guide*.

{ }

{ } does nothing; use it to insert blank lines in the macro without stopping the macro. Quattro Pro continues running the macro command following it.

See { ; } and {STEPON} for examples.

{ ; }

Format {;*String*}

String = numbers or letters up to 1020

{ ; } lets you add explanatory remarks or comments to a macro. When Quattro Pro encounters this macro command, it skips over it.

{ ; } is a convenient way to temporarily hide other macro commands, such as a branch to an incomplete macro.

Examples This example runs the _int_update subroutine, which calculates interest to date, then it branches to the _print_inv subroutine to print the invoice. Notice how the comments embedded in the macro help make it easier to follow.

```
{; calculate interest to date}
{_int_update}
{ }
{; print the invoice}
{BRANCH _print_inv}
```

{ ? }

{ ? } pauses macro execution and lets the macro user access menus, use function keys, and choose commands until *Enter* is pressed. At that point the macro resumes execution.

Use this command with caution; don't include it in macros that might be used by novices.

You must append a {CR} or ~ command after the { ? } to complete any user entry that requires *Enter*. For example, if the macro { ? } was used, and the user typed 67 then pressed *Enter*, the value would remain on the input line at the top of the application window even though the macro resumes. {CR} or ~ (or pressing *Enter* again) would then enter 67 into the cell.

Examples This macro selects cell E15, enters the label `Check Number?` in the cell to act as a prompt, then passes control to the keyboard so the user can enter a check number. The information the user enters replaces the Check Number? prompt.

```
{EditGoto E15}
{PUTCELL "Check Number?"}
{?}~
```

{ABS}

Format {ABS <*Number*>}

Number = controls what part of formula to convert (optional)

{ABS} is equivalent to the Abs key, *F4*. The {ABS} macro converts relative cell addresses to absolute addresses. *Number* provides control over what part of the formula converts to an absolute address. {ABS 1} is equivalent to pressing the Abs key once; {ABS 2} is equivalent to pressing it twice, and so on. If *Number* is missing, {ABS 1} is assumed.

Figure 4.1
How {ABS} affects parts of a cell address

	A:A1	A:$A1	A:$A$1	A:A$1	$A:A1	$A:$A1	$A:$A$1	$A:A$1
{ABS 1}	$A:$A$1	A:A1	A:A$1	A:$A1	A:$A$1	$A:A1	$A:A$1	$A:$A1
{ABS 2}	$A:A$1	$A:$A$1	A:$A1	A:A1	A:A$1	A:$A$1	$A:$A1	$A:A1
{ABS 3}	$A:$A1	$A:A$1	A:A1	$A:$A$1	A:$A1	A:A$1	$A:A1	A:$A$1
{ABS 4}	$A:A1	$A:$A1	$A:$A$1	$A:A$1	A:A1	A:$A1	A:$A$1	A:A$1
{ABS 5}	A:$A$1	$A:A1	$A:A$1	$A:$A1	$A:$A$1	A:A1	A:A$1	A:$A1
{ABS 6}	A:A$1	A:$A$1	$A:$A1	$A:A1	$A:A$1	$A:$A$1	A:$A1	A:A1
{ABS 7}	A:$A1	A:A$1	$A:A1	A:$A$1	$A:$A1	$A:A$1	A:A1	$A:$A$1
{ABS 8}	A:A1	A:$A1	A:$A$1	A:A$1	$A:A1	$A:$A1	$A:$A$1	$A:A$1

{ABS} is most commonly used to create absolute addresses, which are handy for referencing a constant value that appears in only one place in a notebook but is copied in several formulas. The advantage of using absolute addresses is that formulas don't have to be edited if the absolute addresses are created before the formula is copied.

Examples The following example converts the formula in the active cell with a relative cell address to an absolute cell address. Then, it copies the formula to a new location five cells below and five cells to the right of the active cell.

```
\A    {EDIT}{ABS}~
      {BlockCopy C(0)R(0),C(5)R(5)}
```

{ACTIVATE}

Format {ACTIVATE *WindowName*}

WindowName = name of the window to make active

The window's name is on its title bar.

{ACTIVATE} makes the window specified by the string *WindowName* active. For example, to make the named graph PROFITS (in the notebook REPORT.WB1) active, use

```
{ACTIVATE "C:\SALES\REPORT.WB1:PROFITS"}
```

Use the same syntax for activating dialog windows. To make the notebook itself active, use

```
{ACTIVATE "C:\SALES\REPORT.WB1"}
```

See also {CHOOSE} and {WINDOW}.

{ADDMENU}

Format {ADDMENU *MenuPath,MenuBlk*}

MenuPath = location in the menu system to insert a new menu
MenuBlk = block containing a menu definition

{ADDMENU} lets you add menus to the active menu system. (Use {ADDMENUITEM} to add individual menu commands to the active menu system.)

MenuPath is a string that specifies where the new menu should appear. For example, to insert a menu before the Block menu, use `/Block`; to insert a menu before the Copy command on the Block menu, use `/Block/Copy`. You can use <- and -> to place a menu at the top or before the bottom of a menu, respectively. For example, `/File/<-` specifies the last item on the File menu.

You can also use numbers to identify menu commands. For example, `/File/0` specifies the first item on the File menu (the ID numbers start at zero). When identifying a menu item with numbers, divider lines are considered menu items (for example, `/File/5` specifies the first divider line on the File menu, not Retrieve).

MenuBlk is a block containing a menu definition. The block must include all cells in the new menu. See Chapter 7 for more information on menu definitions. See page 372 for an example of {ADDMENU}.

Caution! Changes made to the menu system using this command aren't saved; they're lost when you exit Quattro Pro. Each time you run the macro containing this command, the menu changes are made.

See also {ADDMENUITEM}, {DELETEMENU}, {MENUBRANCH}, and {MENUCALL}.

{ADDMENUITEM}

Format {ADDMENUITEM *MenuPath,Name, <Link>, <Hint>, <HotKey>, <DependString>, <Checked>*}

MenuPath = location in the menu system to insert a new menu item
Name = name of the command to add

Link	= action to perform when the command is chosen (optional)
Hint	= help text to display on the status line when the command is highlighted (optional)
HotKey	= shortcut key that chooses the command (optional)
DependString	= areas in which the command is available (optional)
Checked	= Yes if the command should have a checkmark display by it (optional)

{ADDMENUITEM} is like {ADDMENU}, but inserts a single item before *MenuPath* instead of inserting a new menu. See {ADDMENU} for the syntax of *MenuPath*. *Name* is the name of the new menu command; precede the command's underlined letter with an ampersand (&).

Link is a link command that specifies the actions the menu command performs (for example, `"MACRO _remove_file"` runs the macro _remove_file; see page 376 for more information on link commands). *Hint* is a brief message that appears on the status line when the command is highlighted.

HotKey is a shortcut key the user can press to choose the command (for example, `"Shift+F11"`). *DependString* is a list of Yes/No decisions that specifies where the command is dimmed (see page 378 for the syntax of this string). *Checked* is set to Yes if a checkmark should display by the command.

See Chapter 7 for more information on adding menus and menu commands to the active menu system.

Examples The following macro adds the menu command Erase to the top of the Block menu, and places a divider line between it and Block | Move. Whenever Erase is chosen, the macro _erase_it runs.

```
\A    {ADDMENUITEM "/Block/<-","&Erase","MACRO _erase_it"}
      {ADDMENUITEM "/Block/Move","--------"}
```

Caution! Changes made to the menu system using this command aren't saved; they're lost when you exit Quattro Pro. Each time you run the macro containing this command, the menu changes are made.

See also {ADDMENU} and {DELETEMENUITEM}.

{Align.*Option*}

Command equivalent	Equivalent to ...
{Align.Bottom}	Dialog | Align | Bottom, Draw | Align | Bottom
{Align.Horizontal_Center}	Dialog | Align | Horizontal Center, Draw | Align | Horizontal Center
{Align.Horizontal_Space *Value*}	Dialog | Align | Horizontal Space
{Align.Left}	Dialog | Align | Left, Draw | Align | Left
{Align.Right}	Dialog | Align | Right, Draw | Align | Right
{Align.Top}	Dialog | Align | Top, Draw | Align | Top
{Align.Vertical_Center}	Dialog | Align | Vertical Center, Draw | Align | Vertical Center
{Align.Vertical_Space *Value*}	Dialog | Align | Vertical Space

{Align.*Option*} is the command equivalent for Draw | Align and Dialog | Align; its action depends on whether a graph or dialog window is active.

Value is the amount of space to leave between controls, in pixels. For details, see Chapter 10 of the *User's Guide* (Draw | Align) and Chapter 6 (Dialog | Align).

{ALT}

Format {ALT+*Key* <*Number*>}

Key = keyboard command (PGUP, DOWN, and so on)
Number = number of times to repeat the operation (optional)

{ALT} emulates holding down the *Alt* key while pressing the key specified by *Key*. Uses of {ALT} that would perform Windows task-switching functions (like *Alt+Tab* and *Alt+Esc*) are ignored.

You can combine {ALT}, {CTRL}, and {SHIFT}. For example, {CTRL+SHIFT+D} is equivalent to pressing *Ctrl+Shift+D.*

Examples {ALT+F} emulates pressing *Alt+F,* which could display the File menu or activate a control in a dialog box with an underlined letter of F.

See also {SHIFT} and {CTRL}.

{Application.*Property*}

{Application.*Property*} is equivalent to Property | Application (the application Object Inspector). For details, see Chapter 16 of the *User's Guide*. The next table lists the possible settings for *Property*. Items marked with asterisks (*) only appear in Developer mode (see Chapter 5).

Property	Property|Application option
Current_File	none
Display	Display
Enable_Inspection	Enable Inspection *
International	International
Macro	Macro
SpeedBar	SpeedBar
Startup	Startup
Title	Title *

{Application.Current_File}

Returns the name of the active notebook (only used with @COMMAND).

{Application.Display<.*Option*>}	Equivalent to Property|Application...
{Application.Display "*Clock, SpeedBar, InputLine, Status, BlockSyntax*"}	... | Display
{Application.Display.Clock_Display International | None | Standard}	... | Display | Clock Display
{Application.Display.Range_Syntax "A..B:A1..B2" | "A:A1..B:B2"}	... | Display | 3-D Syntax
{Application.Display.Show_InputLine Yes | No}	... | Display | Show Input Line
{Application.Display.Show_StatusLine Yes | No}	... | Display | Show Status Line
{Application.Display.Show_Toolbar Yes | No}	... | Display | Show SpeedBar

{Application.Display.*Option*} is equivalent to options of the application property Display, which let you specify block syntax and display parts of the Quattro Pro user interface. The arguments of {Application.Display} (which sets all options of the Display property in one command) use the same syntax as those in the {Application.Display.*Option*} commands. For example, the *Clock* argument can be `International`, `None`, or `Standard`, the same settings that {Application.Display.Clock_Display} accepts.

Examples The following macro command hides the time, hides the standard SpeedBar, displays the input line and status line, and sets the block syntax to standard format.

```
{Application.Display "None,No,Yes,Yes,A..B:A1..B2"}
```

{Application.Enable_Inspection Yes|No}

Enables (Yes) or disables (No) Object Inspector menus.

{Application.International<.*Option*>}	Equivalent to Property|Application...
{Application.International "*CurSym, Currency, Placement, Negative, Punctuation, DateFmt, TimeFmt, Language, Conversion*"}	... | International
{Application.International.Currency "Windows Default" | "Quattro Pro/Windows"}	... | International | Currency
{Application.International.Currency_Symbol}	... | International | Currency Symbol
{Application.International.Date_Format *String*}	... | International | Date Format
{Application.International.LICS_Conversion Yes | No}	... | International | Conversion
{Application.International.Negative Signed | Parenthesis}	... | International | Negative Values
{Application.International.Placement Prefix | Suffix}	... | International | Placement
{Application.International.Punctuation "1 234,56 (a1.a2)" | "1 234,56 (a1;a2)" | "1 234.56 (a1,a2)" | "1 234.56 (a1;a2)" | "1,234.56 (a1,a2)" | "1,234.56 (a1;a2)" | "1.234,56 (a1;a2)" | "1.234,56 (a1.a2)" | "Windows Default"}	... | International | Punctuation
{Application.International.Sort_Table *String*}	... | International | Language
{Application.International.Time_Format *String*}	... | International | Time Format

{Application.International.*Option*} is equivalent to the application property International, which lets you specify the punctuation, sort order, and numeric formats used by Quattro Pro. The arguments of {Application.International} (which sets all options of the International property in one command) use the same syntax as those in the {Application.International.*Option*} commands. For example, the *Placement* argument can be `Prefix` or `Suffix`, the same settings that {Application.International.Placement} accepts.

Examples The following macro command sets the currency symbol to $, specifies that the Quattro Pro currency format is used, places the currency symbol before values, sets the punctuation, sets the date and time formats to Windows defaults, sets the sort order to English, and disables LICS conversion. The entire string must be enclosed within a set of quotes, as described on page 121. Individual items within the string that contain spaces or

punctuation must be enclosed in two sets of quotes, as shown in the next example.

Enter all of this example into one cell.

```
{Application.International "$,""Quattro Pro/Windows"",Prefix,
   Parentheses,""1,234.56 (a1,a2)"",""MM/DD/YY (MM/DD)"",
   ""HH:MM:SS (HH:MM)"",""Quattro Pro/Windows"",
   ""English (American)"",No"}
```

{Application.Macro<.*Option*>}	**Equivalent to Property|Application...**
{Application.Macro "*MacSuppress, KeyRead, SlashKey*"}	... | Macro
{Application.Macro.Macro_Redraw Both | None | Panel | Window}	... | Macro | Macro Suppress-Redraw
{Application.Macro.KeyReader Yes | No}	... | Macro | Key Reader
{Application.Macro.Slash_Key *MenuName*}	... | Macro | Slash Key

{Application.Macro.*Option*} is equivalent to options of the application property Macro, which let you run 1-2-3 macros, control screen updates, and display alternative menu systems. The arguments of {Application.Macro} (which sets all options of the Macro property in one command) use the same syntax as those in the {Application.Macro.*Option*} commands. For example, the *KeyRead* argument can be `Yes` or `No`, the same settings that {Application.Macro.KeyReader} accepts.

Examples The following macro command specifies that windows shouldn't display when a macro runs, disables 1-2-3 macros, and makes the slash key display the Quattro Pro for DOS menu system.

```
{Application.Macro "Window,No,Quattro Pro - DOS"}
```

{Application.SpeedBar *SpeedBarName*}

Displays or removes a custom SpeedBar (below the standard Quattro Pro SpeedBar). To remove the custom SpeedBar, use `{Application.SpeedBar ""}`.

{Application.Startup<.*Option*>}	**Equivalent to Property|Application...**
{Application.Startup "*StartDir, AutoFile, StartMacro, FileExt,UseBeep, UseUndo, CompatibleKeys*"}	... | Startup
{Application.Startup.Autoload_File *String*}	... | Startup | Autoload File
{Application.Startup.Beep Yes | No}	... | Startup | Use Beep
{Application.Startup.Compatible_Keys Yes | No}	... | Startup | Compatible Keys

{Application.Startup.File_Extension *String*} ... | Startup | File Extension
{Application.Startup.Startup_Directory *String*} ... | Startup | Directory
{Application.Startup.Startup_Macro *String*} ... | Startup | Startup Macro
{Application.Startup.Undo Yes | No} ... | Startup | Undo Enabled

{Application.Startup.*Option*} is equivalent to the application property Startup, which lets you specify startup options for Quattro Pro like the autoload macro name and the initial directory to display in file controls. The arguments of {Application.Startup} (which sets all options of the Startup property in one command) use the same syntax as those in the {Application.Startup.*Option*} commands. For example, the *UseBeep* argument can be `Yes` or `No`, the same settings that {Application.Startup.Beep} accepts.

Examples The following macro command sets the starting directory to C:\FILES, sets the autoload file to START.WB1, sets the autoload macro to Init, sets the default file extension to WB1, disables the Quattro Pro beep, enables Undo, and disables compatible keys.

```
{Application.Startup "C:\FILES,START.WB1,Init,WB1,No,Yes,No"}
```

{Application.Title *Title*}

Sets the title appearing in the Quattro Pro title bar.

{BACKSPACE} and {BS}

Format {BACKSPACE <*Number*>}

Number = any positive integer or the address of a cell containing a positive integer (optional)

{BACKSPACE} and {BS} are equivalent to the *Backspace* key, which deletes one character to the left of the insertion point in Edit mode. The optional argument *Number* specifies how many times to repeat the operation; for example, {BACKSPACE 2} is equivalent to pressing *Backspace* twice.

{BACKTAB}

Format {BACKTAB <*Number*>}

Number = any positive integer or the address of a cell containing a positive integer (optional)

{BACKTAB} is equivalent to the *Ctrl* ← or *Shift* + *Tab* key. It's the same as {BIGLEFT}, which selects the leftmost cell of the screen to the left of the current one. The optional argument *Number* specifies how many times to repeat the operation; for example, {BACKTAB 3} is equivalent to pressing *Shift+Tab* three times.

{BEEP}

Format {BEEP <*Number*>}

Number = 1 to 10 (optional)

{BEEP} sounds the computer's built-in speaker. *Number* dictates the tone of the beep. If *Number* is omitted, {BEEP 1} sounds. If *Number* is larger than 10, the pattern repeats; for example, {BEEP 11} is the same as {BEEP 1}.

Use {BEEP} to catch the user's attention. You can use it in interactive macros to introduce a prompt for information or to indicate a macro has finished.

Examples The following macro checks a block named error_check for an error condition (indicated by error_check containing a number other than zero). If there's no error, it branches to a macro called _continue, which carries on the previous procedure. If there is an error, it gives a low beep, then a medium beep, and moves the selector to the block called message_area, where an error message is stored.

```
{IF error_check = 0}{BRANCH _continue}
{BEEP 1}{BEEP 5}{EditGoto message_area}
```

{BIGLEFT}

Format {BIGLEFT <*Number*>}

Number = any positive integer or the address of a cell containing a positive integer (optional)

{BIGLEFT} is equivalent to the *Ctrl*← or *Shift+Tab* key. It's the same as {BACKTAB}, which selects the leftmost cell of the screen that's to the left of the current one. The optional argument *Number* specifies how many times to repeat the operation; for example, {BIGLEFT 2} is equivalent to pressing *Shift+Tab* twice.

{BIGRIGHT}

Format {BIGRIGHT <*Number*>}

Number = any positive integer or the address of a cell containing a positive integer (optional)

{BIGRIGHT} is equivalent to the *Ctrl*→ or *Tab* key. It's the same as {TAB} and selects the leftmost cell of the screen that's to the right of the current one. The optional argument *Number* specifies how many times to repeat the operation; for example, {BIGRIGHT 2} is equivalent to pressing *Tab* twice.

{BLANK}

Format {BLANK *Location*}

Location = cell or block you want erased

{BLANK} erases the contents of the block referred to as *Location*. You can also use the command equivalents {ClearContents} and {EditClear} to erase the contents of the currently selected block.

Examples The following macro erases the block named part_list.

```
\F    {BLANK part_list}
```

{BlockCopy}

Format {BlockCopy *SourceBlock, DestBlock,* <*ModelCopy?*(0 | 1)>}

SourceBlock = block to copy
DestBlock = location to copy block
ModelCopy? = whether to use Model Copy option; 0 = no, 1 = yes

{BlockCopy} is the command equivalent for Block | Copy. It copies the source block to the specified destination. If *ModelCopy?* is 1, absolute references to cells within the copied block adjust to reflect the new location. For details, see Chapter 3 of the *User's Guide.*

{BlockDelete.*Option*}

Command equivalent	Equivalent to Block|Delete...
{BlockDelete.Columns *Block*, Entire | Partial}	... | Columns
{BlockDelete.Pages *Block*, Entire | Partial}	... | Pages
{BlockDelete.Rows *Block*, Entire | Partial}	... | Rows

{BlockDelete.*Option*} is the command equivalent for Block | Delete. It deletes entire or partial columns, rows, and pages. For details, see Chapter 3 of the *User's Guide. Block* is the 2-D or 3-D block where material is deleted.

{BlockFill.*Option*}

Command equivalent	Equivalent to Block|Fill...
{BlockFill.Block *Block*}	... | Blocks
{BlockFill.Go}	... | OK
{BlockFill.Order Column | Row}	... | Fill | Order
{BlockFill.Series Linear | Growth | Power | Year | Month | Week | Weekday | Day | Hour | Minute | Second}	... | Fill | Series
{BlockFill.Start *Value*}	... | Fill | Start
{BlockFill.Step *Value*}	... | Fill | Step
{BlockFill.Stop *Value*}	... | Fill | Stop

{BlockFill.*Option*} is the command equivalent for Block | Fill. It fills *Block* with sequential data. You can use numbers, dates, times, or even formulas for *Value*. For details, see Chapter 2 of the *User's Guide.*

- If {BlockFill.Start} is a number or formula, you can enter one of these strings for {BlockFill.Series}:
  - "Linear" adds the step value to the previous value (defined at first to be the start value).
  - "Growth" multiplies the step value by the previous value.

- "Power" uses the step value as the exponent of the previous value.

- If {BlockFill.Start} is a date or time, the fill operation is always linear, but you can specify the step unit as "Year", "Month", "Week", "Weekday", "Day", "Hour", "Minute", or "Second". For example, with a start value of 6/20/92, a step value of 2, and "Month" as the {BlockFill.Series *Option*} setting, the second cell in the filled block contains `August`.

  You can enter the date and time directly as a serial number or use one of the date and time @functions (see page 17).

Examples The following macro uses @DATEVALUE to enter 6/20/92 as the start value. The 3-D block to fill is B..C:B1..D4 with a step value of 2. Fill order is "Row".

```
{BlockFill.Block B:B1..C:D4}
{BlockFill.Start @DATEVALUE("6/20/92")}
{BlockFill.Step 2}
{BlockFill.Stop @DATEVALUE("12/31/2099")}
{BlockFill.Order Row}
{BlockFill.Series Month}
{BlockFill.Go}
```

{BlockInsert.*Option*}

Command equivalent	Equivalent to Block|Insert...
{BlockInsert.Columns *Block*, Entire | Partial}	... | Columns
{BlockInsert.File *FileName*, *BeforeBlock*}	... | File
{BlockInsert.Pages *Block*, Entire | Partial}	... | Pages
{BlockInsert.Rows *Block*, Entire | Partial}	... | Rows

{BlockInsert.*Option*} is the command equivalent for Block | Insert. It inserts entire or partial columns, rows, and pages, or complete files. For details, see Chapter 3 of the *User's Guide*. *Block* is the 2-D or 3-D block where material is inserted.

{BlockMove}

Format {BlockMove *SourceBlock*, *DestBlock*}

SourceBlock = block to move
DestBlock = new location for the block

{BlockMove} is the command equivalent for Block | Move. It moves the contents of a block from one location to another. Any data already in *DestBlock* is overwritten by the contents of *SourceBlock*. Block properties move with the data. Nonabsolute formulas are adjusted.

For details, see Chapter 3 of the *User's Guide*. To move pages, use {BlockMovePages}, described next.

{BlockMovePages}

Format {BlockMovePages *SrcPages, BeforePage*}

SrcPages = range of pages to move
BeforePage = new location for *SrcPages*

{BlockMovePages} is the command equivalent for Block | Move Pages. It reorders pages within a notebook. Moved pages appear before *BeforePage*.

For details, see Chapter 3 of the *User's Guide*.

{BlockName.*Option*}

Command equivalent	Equivalent to Block|Names...
{BlockName.Create *BlockName, Block*}	... | Create
{BlockName.Delete *BlockName*}	... | Delete
{BlockName.Labels Left | Right | Up | Down}	... | Labels
{BlockName.MakeTable *Block*}	... | Make Table
{BlockName.Reset}	... | Reset

{BlockName.*Option*} is the command equivalent for Block | Names. It creates, deletes, and displays names for contiguous and noncontiguous blocks. For details, see Chapters 3 and 11 of the *User's Guide*.

BlockName is the block name to create or delete. In {BlockName.Create}, *Block* is the block to name; in {BlockName.MakeTable}, *Block* indicates where to create the name table. {BlockName.Reset} clears all block names in the notebook. If you omit the arguments for {BlockName.Create}, {BlockName.Delete}, or {BlockName.MakeTable}, the appropriate dialog box appears.

{BlockReformat}

Format {BlockReformat *Block*}

Block = the block to reformat

{BlockReformat} is the command equivalent for Block | Reformat. It adjusts word wrapping in a series of label entries as though they were in a paragraph. For details, see Chapter 11 of the *User's Guide.*

{BlockTranspose}

Format {BlockTranspose *SourceBlock, DestBlock*}

SourceBlock = block to transpose
DestBlock = block to hold transposed copy

{BlockTranspose} is the command equivalent for Block | Transpose. It copies a block to another location and reverses its rows and columns. Existing data in *DestBlock* is overwritten. For details, see Chapter 11 of the *User's Guide.*

{BlockValues}

Format {BlockValues *SourceBlock, DestBlock*}

SourceBlock = block to copy as values
DestBlock = block to hold converted copy

{BlockValues} is the command equivalent for Block | Values. It copies a block to another location and converts its formulas to values. Existing data in *DestBlock* is overwritten. For details, see Chapter 11 of the *User's Guide.*

{BRANCH}

Format {BRANCH *Location*}

Location = location or name of another macro

{BRANCH} runs the macro stored in *Location*. If *Location* references a block, Quattro Pro starts with the macro command in the top left cell.

{BRANCH} can change the flow of execution based on a condition test. For example, you can use it to run a different macro depending on the contents of a certain cell.

{BRANCH} is like {Subroutine} in that it passes control to another macro. Unlike {Subroutine}, it doesn't hold your place in the original macro, waiting for control to return. Use {BRANCH} when you don't intend to return to the original macro. To run another macro and then return to the calling macro, use {Subroutine}. (See page 269 for information on subroutines.)

Examples The following macro branches to a macro named _high if the value in cell D10 is greater than 1000; otherwise, it continues to run the macro on the next line.

```
{IF D10 > 1000}{BRANCH _high}
```

{BREAK}

{BREAK} clears any displayed dialog boxes or prompts and returns Quattro Pro to Ready mode. It doesn't stop macro execution; you can use {QUIT} to stop a macro.

{BREAKOFF}

{BREAKOFF} disables *Ctrl+Break*, which can be used to end a macro before it's done. After {BREAKOFF}, the user won't be able to exit a macro until the end of the macro or until a {BREAKON} command is used.

Caution! Use {BREAKOFF} only when necessary. Without access to *Ctrl+Break*, the only way to stop a "runaway" macro is to reboot or turn off the computer.

Examples The following macro disables *Ctrl+Break* while the user inputs a name.

```
{PUTCELL "Enter your name here:"}
{BREAKOFF}
{?}~
{BREAKON}
{PUTCELL "Try again: "}
{?}~
```

{BREAKON}

{BREAKON} enables *Ctrl+Break* after a previous {BREAKOFF} command has disabled it.

{BREAKON} should be used as soon as possible after {BREAKOFF}, because with *Ctrl+Break* disabled, the only way to halt a "runaway" macro is to reboot or turn off the computer.

See {BREAKOFF} for an example of {BREAKOFF} and {BREAKON}.

{CALC}

{CALC} is equivalent to the Calc key, *F9*, which recalculates the active notebook or converts the formula on the input line into its result when editing a cell.

{CAPOFF} and {CAPON}

{CAPOFF} and {CAPON} are equivalent to *Caps Lock* off and *Caps Lock* on, respectively.

{CHOOSE}

{CHOOSE} displays a pick list of open windows. Your choice becomes the active window.

See also {ACTIVATE}.

{CLEAR}

{CLEAR} is the equivalent of *Ctrl+Backspace*, which erases any previous entry in a prompt line or on the input line in Edit mode. It is useful when loading and retrieving files.

{ClearContents}

Format {ClearContents <*fPageOnly?* (0 | 1)>}

fPageOnly? = whether to operate on only the active page of a group (1) or on all pages (0); only applies in Group mode

{ClearContents} is the command equivalent for Edit | Clear Contents. It erases the contents of the selected block but leaves block property settings intact. For details, see Chapter 3 of the *User's Guide*.

{CLOSE}

{CLOSE} ends access to a file previously opened with {OPEN}. This lets another file be opened (only one can be open at a time). {CLOSE} completes the process of writing information to a file, including an update of the disk directory. This step is crucial to the integrity of any file. If your computer is turned off before a file is closed, that file's contents may become corrupted or lost.

{CLOSE} fails in the event of a disk error, such as when a disk is removed from the disk drive before the file is closed. In this case, {ONERROR} is useful in intercepting the error. If {CLOSE} succeeds, macro execution continues in the cell below the cell containing the {CLOSE} command, ignoring any other commands in that cell. If {CLOSE} fails, macro execution continues in the same cell as the {CLOSE} command.

Examples The following macro opens a new file in drive A called AFILE, writes the ASCII text line `Hello, world!` to the file, and closes the file.

```
\F    {OPEN "A:\AFILE",W}
      {WRITELN "Hello, world!"}
      {CLOSE}
```

{COLUMNWIDTH}

Format {COLUMNWIDTH *Block, FirstPane?,Set/Reset/Auto, Size*}

Block = block containing columns to resize

FirstPane?	= 1 to resize columns in left or top window pane; 0 to resize columns in right or bottom window pane
Set/Reset/Auto	= 0 to set the column width; 1 to reset the column width; 2 to automatically size the column(s)
Size	= new width (in twips) if *Set/...* = 0; not needed if *Set/...* = 1; resetting size; extra characters (optional) if *Set/...* = 2

{COLUMNWIDTH} provides three ways to change the width of a column or block of columns (it is equivalent to the block property Column Width). The columns to change are specified by *Block*. *FirstPane?* is used when the active window is split into panes (using Window | Panes). To resize the columns in the left or top pane, set *FirstPane?* to 1; to resize the columns in the right or bottom pane, set *FirstPane?* to 0.

The argument *Set/Resize/Auto* specifies how to change the width. To set a column width, use the following syntax:

{COLUMNWIDTH *Block, FirstPane?*, 0, *NewSize*}

NewSize is the new column width, in twips (a twip is 1/1440th of an inch). The maximum width is 20 inches (28,800 twips).

To reset a column to the default width (set by Default Width in the page Object Inspector) use the following syntax:

{COLUMNWIDTH *Block, FirstPane?*, 1}

To automatically size a column based on what's entered in it, use the following syntax:

{COLUMNWIDTH *Block, FirstPane?*, 2, *ExtraCharacters*}

ExtraCharacters is the number of characters to add on to the calculated width. If this argument is omitted, the default is used (1 character).

Examples

`{COLUMNWIDTH A:A..B,1,0,1440}` sets the width of columns A and B (on page A) to one inch (1,440 twips).

`{COLUMNWIDTH A:A..B,0,0,2160}` sets the width of columns A and B (on page A) to one and a half inches (2,160 twips). If the window is split, the columns are resized in the left or top pane.

`{COLUMNWIDTH A:C,1,1}` resets the width of column C (on page A) to the default width.

`{COLUMNWIDTH A:C,1,2,3}` automatically sizes column C (on page A) and adds three characters to the calculated width.

See also {ROWHEIGHT} and {ROWCOLSHOW}.

{CONTENTS}

Format {CONTENTS *Dest,Source,<Width#,Format#>*}

Dest = cell you want data written to
Source = cell you want data copied from
Width# = optional column width (1 to 1023)
Format# = optional format code (see Table 4.13)

{CONTENTS} copies the contents of *Source* into *Dest*, but unlike {LET} or other copy commands, if *Source* contains a value entry, {CONTENTS} translates the copied value into a label and stores it in *Dest* (in the source cell's numeric format). It also lets you specify a different numeric format and column width using the *Width#* and *Format#* arguments.

Width# can be any number from 1 to 1023. Quattro Pro won't alter the width of the destination column but will treat the resulting string as if it came from a column with the specified width. For example, if a value is displayed as ***** in the source column because the column isn't wide enough, specifying a wider *Width#* will let the value be copied as it would be displayed within that width, not as *****. *Width#* is optional, but must be provided if *Format#* is used. If you don't specify *Width#*, the width of the source column is assumed. Use the maximum width if you want all values to come across properly. You can use @TRIM with a {LET} command to remove any leading spaces from the label.

Format# can be any number from 0 to 127. Each number in this range corresponds to a specific numeric format and decimal precision (see Table 4.13). *Format#* affects the *Dest* entry only, not the *Source* value.

The following table lists the special codes used to indicate numeric formats with *Format#*.

Table 4.13
Numeric format codes

Code	Description
0-15	Fixed (0-15 decimals)
16-31	Scientific (0-15 decimals)
32-47	Currency (0-15 decimals)
48-63	% (percent; 0-15 decimals)
64-79	, (comma; 0-15 decimals)
112	+/- (bar graph)
113	General
114	Date [1] (DD-MMM-YYYY)
115	Date [2] (DD-MMM)
116	Date [3] (MMM-YYYY)
117	Text
118	Hidden
119	Time [1] (HH:MM:SS AM/PM)
120	Time [2] (HH:MM AM/PM)
121	Date [4] (Long International)
122	Date [5] (Short International)
123	Time [3] (Long International)
124	Time [4] (Short International)
127	Default (set with Normal style)

Examples The following examples assume cell C18 contains the value 48,988 in comma format with a column width of 12 characters.

`{CONTENTS A18,C18}` places the 12-character label '   48,988 in cell A18 (six spaces are inserted at the beginning).

`{CONTENTS E10,C18,3}` places the 3-character label '*** in cell E10. (Only asterisks are copied because the value doesn't fit within three spaces.)

`{CONTENTS A5,C18,15,34}` places the 15-character label ' $48,988.00 in cell A5 (five spaces are inserted at the beginning).

{Controls.*Option*}

Command equivalent	Equivalent to Dialog|Order...
{Controls.Order}	... | Order Controls
{Controls.OrderFrom}	... | Order From
{Controls.OrderTab}	... | Order Tab Controls
{Controls.OrderTabFrom}	... | Order Tab From

{Controls.*Option*} is equivalent to Dialog | Order. It affects selected objects in the dialog window. For details, see Chapter 6.

{CR} or ~

{CR} or ~ (tilde) are equivalent to the *Enter* key.

{CREATEOBJECT}

Format {CREATEOBJECT *ObjectName, x1, y1, x2, y2, <x3, y3>...*}

ObjectName = type of object to create

x1, y1 = *xy* coordinates for the starting point of the object in pixels; the upper left corner for rectangles and objects bounded by rectangles

x2, y2 = *xy* coordinates for the end point or next point of the object in pixels; the width and height for rectangles and objects bounded by rectangles

x3, y3 = *xy* coordinates for the next or last point of a polyline or polygon object

With {CREATEOBJECT} you can add objects to the active window normally added using the SpeedBar. {CREATEOBJECT} is context sensitive, letting you create lines in a graph window or check boxes in a dialog window, for example. Commands out of context are ignored. Quattro Pro interprets the coordinates specified after *ObjectName* differently for each object type. The following table lists the possible graph object settings for *ObjectName*. The paragraph following the table lists dialog object settings.

Table 4.14: Graph objects {CREATEOBJECT} can generate

Object	# of (x,y)'s	Coordinates
Line	2	1st: Start point; 2nd: End point
Arrow	*(same as for Line)*	
Rect (Rectangle)	2	1st: Upper left corner; 2nd: width and height
Ellipse	2	1st: Upper left corner of a rectangle bounding the ellipse; 2nd: width and height of the bounding rectangle
RoundedRectangle	*(same as for Rectangle)*	
Text	*(same as for Rectangle)*	
Polyline	Varies	1st: Start point; 2nd: End point of first segment and start of second segment; 3rd: End point of second segment and start of third segment, ... *n*th: End point
Polygon	*(same as for Polyline)*	
FreehandPolyline	*(same as for Polyline)*	
FreehandPolygon	*(same as for Polyline)*	

You can create these dialog controls, listed in the order they appear on the Dialog SpeedBar: Button, CheckBox, RadioButton, BitmapButton, Label, EditField, SpinCtrl, Rectangle, GroupBox, RangeBox, ComboBox, PickList, FileCtrl, ColCtrl, ScrollBar, HScrollBar, TimeCtrl. When creating a control, *x1* and *y1* specify the upper left corner of the control; *x2* and *y2* specify the width and height of the control in pixels.

ObjectName is enclosed in quotes. The x and y coordinates for each point follow, separated by commas.

Examples `{CREATEOBJECT "Rectangle", 86,11,94,74}` creates a rectangle whose upper left corner is at (86,11), width is 94 pixels, and height is 74 pixels.

`{CREATEOBJECT "Line", 260,238,356,228}` creates a line that starts at (260,238) and ends at (356,228).

`{CREATEOBJECT "Polyline", 2,2,23,59,11,26}` creates a polyline that starts at (2,2), draws a line to (23,59), and then draws a line from that point to (11,26).

See also {MOVETO} and {RESIZE}.

{CTRL}

Format {CTRL+*Key* <*Number*>}

Key = keyboard macro command (PGUP, DOWN, and so on)
Number = number of times to repeat the operation (optional)

{CTRL} emulates holding down the *Ctrl* key while pressing a keystroke. For example, `{CTRL+PGDN 5}` emulates pressing *Ctrl+PgDn* five times to move down five notebook pages. Keystrokes that would perform Windows task switching, such as *Ctrl+Esc*, are ignored.

See also {ALT} and {SHIFT}.

{DATE}

{DATE} is equivalent to the *Ctrl+Shift+D* key, which lets the user enter a date or time into the active cell.

Examples `{DATE}8/6/90~` enters 8/6/90 in the active cell as a date.

`{DATE}{?}~` pauses to let the user enter a date, then enters that date into the active cell.

{DEFINE}

Format {DEFINE *Location1*<:*Type1*>,*Location2*<:*Type2*>,...}

Location = cell in which you want to store the argument being passed

Type = `string` or `value`; `string` (or just `s`) defines the argument as a label, and `value` (or just `v`) defines it as an actual value or value resulting from a formula (optional)

When you pass control to a subroutine with the {Subroutine} command, you can also pass arguments for use by that subroutine. If you do so, you must include a {DEFINE} command in the subroutine's first line. This command defines the data type of each argument passed and indicates which cells to store the arguments in. If no {DEFINE} command is included, the arguments are ignored.

{DEFINE} sequentially defines the arguments passed to the subroutine. The first location and type given are assigned to the first argument passed, the second location and type to the second argument, and so on.

You must specify a location for each argument. This tells Quattro Pro where to copy them. If there are more locations given than arguments passed, or more arguments than locations, the macro ends immediately, and an error message appears.

Type is optional. It tells Quattro Pro whether the argument is a value or string. If no data type is given, the argument is assumed to be a literal string (even if it's a valid block name, cell address, or value). If you add `:string` (or `:s`) to the location, any argument passed is stored as a label. If you add `:value` (or `:v`), Quattro Pro treats the coming argument as a number or number resulting from a formula. If it isn't, Quattro Pro treats it as a string (or string value from a formula).

Examples In the following example, the \F macro passes three arguments (principal, interest, and term) to the subroutine _calc_loan, which stores the arguments in named blocks and defines them as values.

It then

- uses the arguments to calculate the monthly payment on a loan
- stores the result in a cell named amount
- creates a label in a cell named payment displaying that amount as currency
- returns control to the main macro

The main macro displays the result in the active cell, preceded by the string "The monthly payment will be ".

```
\F           {_calc_loan 79500,12%,30}

_calc_loan   {DEFINE prin:value,int:value,term:value}
             {LET amount,@PMT(prin,int/12,term*12)}
             {CONTENTS payment,amount,9,34}
             {EditGoto txt_area}
             {RETURN}

prin         79500
int          0.12
term         30

amount       817.747
payment      $817.75

txt_area     +"The monthly payment will be "&@TRIM(payment)
```

{DEL} and {DELETE}

{DEL} and {DELETE} are equivalent to the *Del* key.

{DELETEMENU}

Format {DELETEMENU *MenuPath*}

MenuPath = menu in the tree to delete

{DELETEMENU} removes the menu specified by *MenuPath* from the menu system. See the description of {ADDMENU} for the syntax of *MenuPath*. Use {DELETEMENUITEM} to remove an individual menu command.

Examples `{DELETEMENU "/File"}` removes the File menu from the active menu system.

Caution! Changes made to the menu system using this command aren't saved; they're lost when you exit Quattro Pro. Each time you run the macro containing this command, the menu changes are made.

See also {ADDMENU} and {DELETEMENUITEM}.

{DELETEMENUITEM}

Format {DELETEMENUITEM *MenuPath*}

MenuPath = menu item in the tree to delete

{DELETEMENUITEM} removes the menu command specified by *MenuPath* from the menu system. Use {DELETEMENU} to remove entire menus from the active menu system. See the description of {ADDMENU} for the syntax of *MenuPath*.

Examples `{DELETEMENUITEM "/Edit/Clear"}` removes the Edit | Clear command.

`{DELETEMENUITEM "/Edit/<-"}` removes the first item on the Edit menu.

Caution! Changes made to the menu system using this command aren't saved; they're lost when you exit Quattro Pro. Each time you run the macro containing this command, the menu changes are made.

See also {ADDMENUITEM}.

{DialogView}

Format {DialogView *Window*}

Window = dialog window to make active

{DialogView} lets you edit an existing dialog box. *Window* is the name of the dialog box to edit.

{DialogWindow.*Property*}

{DialogWindow.*Property*} is equivalent to Property | Dialog (the dialog window Object Inspector). For details, see Chapter 6.

{DialogWindow} commands affect the active dialog window. The next table lists the possible settings for *Property*.

Property	Property|Dialog option
Dimension	Dimension
Disabled	Disabled
Grid_Options	Grid Options
Name	Name
Position_Adjust	Position Adjust
Title	Title
Value	Value

{DialogWindow.Dimension<.*Option*>}	Equivalent to Property|Dialog...
{DialogWindow.Dimension "*XPos,YPos,Width,Height*"	... | Dimension
{DialogWindow.Dimension.Height *Height*}	... | Dimension | Height
{DialogWindow.Dimension.Width *Width*}	... | Dimension | Width
{DialogWindow.Dimension.X *XPos*}	... | Dimension | X Pos
{DialogWindow.Dimension.Y *YPos*}	... | Dimension | Y Pos

{DialogWindow.Dimension.*Option*} is equivalent to the dialog window property Dimension, which lets you move and resize the active dialog window. Each argument is specified in pixels. *XPos* and *YPos* specify the distance in pixels from the left side of the Quattro Pro window and bottom of the input line, respectively.

Examples The following macro command positions the active dialog window two pixels from the left edge of the Quattro Pro window, five pixels below the input line, sets the width to 150 pixels, and sets the height to 250 pixels.

```
{DialogWindow.Dimension "2,5,150,250"}
```

{DialogWindow.Disabled Yes|No}

Disables (Yes) or enables (No) the active dialog box or SpeedBar. This command only works when the user is viewing a dialog box or SpeedBar; it doesn't work when editing one.

{DialogWindow.Grid_Options *GridSize,ShowGrid,SnapToGrid*}

Sets the grid size of the active dialog window. Use *GridSize* to specify the distance between grid points, in pixels; *ShowGrid* specifies whether the grid is visible; *SnapToGrid* specifies whether the grid is active.

Examples The following macro sets the distance between grid points to 10, hides the grid, and enables it.

```
{DialogWindow.Grid_Options "10,No,Yes"}
```

{DialogWindow.Name *Name*}

Sets the name of the active dialog window. This name is used by macro commands, @functions, and link commands to identify the dialog box (or SpeedBar).

{DialogWindow.Position_Adjust *Depend, LeftRel,TopRel, RightRel,BottomRel,CenterHor,CenterVer*}

Specifies how the active dialog box resizes when the Quattro Pro window is resized. See Chapter 6 for details.

{DialogWindow.Title *String*}

Specifies the title that appears on the dialog box when the user is viewing it (the title doesn't appear when editing the dialog box).

{DialogWindow.Value *String*}

Lets you set the initial settings of the dialog box (or SpeedBar). You can use it with @COMMAND to find the current settings of the dialog box. *String* is a comma separated list of settings. Each setting sets the initial value of one control. Control values appear in this list if their Process Value property is set to Yes. You can set the order of the settings while editing the dialog box (see page 330 for details on ordering controls).

Examples The following macro command sets the initial values of a dialog box with three controls. Each setting maps to one control. You can use this command to set the initial values of the dialog box LOANDB (see Chapter 6 for details on the dialog box LOANDB).

```
{DialogWindow.Value "25000,5,1st of month"}
```

{DISPATCH}

Format {DISPATCH *Location*}

Location = a single cell containing the address or block name of another macro

{DISPATCH} is similar to {BRANCH}, except it uses *Location* differently:

- If *Location* is a cell address or single-cell named block, {DISPATCH} branches to the address *stored* in that cell.
- If *Location* is a text formula resulting in a cell or single-cell named block, {DISPATCH} branches to that result address.
- If *Location* is a multi-celled block or a text formula resulting in a multi-celled block, {DISPATCH} branches to the first cell of that block.

{DISPATCH} is useful when you want to branch to one of several alternative macros, depending on circumstances. You can also set up a macro that modifies the contents of *Location*, depending on user input or test conditions.

Examples The macro _continue in the following example uses {DISPATCH jump_cell} to create a form letter that changes depending on the result of a credit check.

Type the line to the right of \F with no hard returns until after the first {BRANCH _continue}, then press Enter to insert the text into one cell. We've had to break it up to print it on the page.

```
credit      OK

\F          {EditGoto text_area}
            {IF credit="OK"}{LET jump_cell, "_ok_note"}{BRANCH
            _continue}
            {LET jump_cell,"_bum_note"}{BRANCH _continue}

_continue   {PUTCELL "Because of your standing with P.K."}
            {DOWN}
            {PUTCELL "Finance, we have decided to"}{DOWN}
            {DISPATCH +jump_cell}

jump_cell   _ok_note

_ok_note    {PUTCELL "extend your credit limit."}{DOWN 2}
            {BRANCH _finish}

_bum_note   {PUTCELL "CANCEL your account."}{DOWN 2}
            {BRANCH _finish}

_finish     {PUTCELL "Please call our office for details."}
            {DOWN 2}
```

```
              {PUTCELL "Sincerely,"}
              {DOWN 3}
              {PUTCELL "James Madison (Account Officer)"}
              {DOWN}
text_area     Because of your standing with P.K.
              Finance, we have decided to
              extend your credit limit.

              Please call our office for details.

              Sincerely,


              James Madison (Account Officer)
```

{DODIALOG}

Format {DODIALOG *DialogName, OKExit?, <Arguments>, <MacUse?>*}

DialogName	= name of dialog box to display
OKExit?	= cell to store how the dialog box closed (1 for OK, 0 for Cancel)
Arguments	= block to store the initial settings for controls in the dialog box and their final settings (optional)
MacUse?	= 1 if the user should manipulate the dialog box, 0 if the macro manipulates it (optional; 1 is default)

{DODIALOG} displays a dialog box for the user or the macro to manipulate (see Chapter 6 for details on creating these dialog boxes). *MacUse?* specifies how the dialog box is manipulated: if 0, the macro manipulates the dialog box; if 1, the macro pauses so the user can manipulate the dialog box. *DialogName* is the name of the dialog box to display. *Arguments* is a block containing initial settings for controls in the dialog box. Each cell in *Arguments* corresponds to one control in the dialog box with a Process Value property set to Yes; the order of controls in the dialog box determines which cell maps to which control (first cell maps to the first control, second cell maps to the second, and so on). See page 330 for a discussion of ordering controls and the Process Value property.

If a dialog box is under macro control (*MacUse?* set to 0), you can make it revert to user control using {PAUSEMACRO}.

Once the dialog box closes (performing its function), Quattro Pro writes the new control settings into their respective cells. *OKExit?* is a cell containing the result of the dialog box operation; 1 if OK was chosen, 0 if the dialog box was canceled.

Examples See page 331 for a macro that uses {DODIALOG} to display a dialog box.

See also {GETLABEL}, {GETNUMBER}, and {PAUSEMACRO}.

{DOWN} and {D}

Format {DOWN <*Number*>}

Number = any positive integer (optional)

{DOWN} and {D} are equivalent to the ↓ key. The optional argument *Number* moves the selector down the corresponding number of rows. You can also use cell references or block names as arguments.

Examples `{DOWN 4}` moves the selector down four rows.
`{DOWN G13}` moves down the number of rows specified in cell G13.
`{DOWN count}` moves down the number of rows specified in the single-cell block named *count*.

{EDIT}

{EDIT} is equivalent to the Edit key, *F2*. Its main use is in Edit mode, where it lets you edit the contents of the active cell. You can also use it to search for items in a long list.

{EditClear}

{EditClear} is the command equivalent for Edit | Clear. It erases the contents and properties of the current block, deletes selected objects from dialog and graph windows, and deletes selected floating objects. To erase a block while leaving its properties intact, use {ClearContents}. For details, see Chapter 3 of the *User's Guide*.

{EditCopy}

{EditCopy} is the command equivalent for Edit | Copy. It copies the selected object to the Clipboard.

{EditCut}

{EditCut} is the command equivalent for Edit | Cut. It deletes the selected object and moves it to the Clipboard.

{EditGoto}

Format {EditGoto *Block*}

Block = block to display and select

{EditGoto} is the command equivalent for Edit | Goto. It selects and displays *Block* within spreadsheet pages, not the Graphs page.

{EditPaste}

{EditPaste} is the command equivalent for Edit | Paste. It copies data and its properties from the Clipboard into the notebook. For details, see Chapter 11 of the *User's Guide*.

To paste only values or properties, use {PasteSpecial}. {PasteLink} creates a live DDE link, and {PasteFormat} adds many types of data from other applications (including embedded OLE objects).

{END}

{END} is equivalent to the *End* key.

{ESC} and {ESCAPE}

{ESC} and {ESCAPE} are equivalent to the *Esc* key, which is useful for clearing menus or dialog boxes from the screen, one at a time. To clear all prompts and return Quattro Pro to Ready Mode, use {BREAK}.

{EXEC}

Format {EXEC *AppName, WindowMode<, ResultLoc>*}

AppName = name (in quotes) of the application to run (up to 100 characters)
WindowMode = size state of the application's window: 1 for normal size, 2 for minimized, 3 for maximized
ResultLoc = a cell which contains the coded explanation of the operation's success

{EXEC} lets you run other Windows applications or DOS commands. *AppName* can be any valid command string you could type in the Program Manager's File | Run command. *AppName* can contain the path of the application. If no path is given (for example, `"VISION.EXE"`), then the application must be in the system path so Windows can find it.

ResultLoc is useful in some cases where the started application supports DDE along with an instance number. For example, Excel and Quattro Pro respond to their names and also respond to their names concatenated with the instance number.

Examples `{EXEC "NOTEPAD.EXE",1}` runs the Windows Notepad and displays it just as if it were run from the Program Manager.

`{EXEC "VISION.EXE",2}` runs ObjectVision, and places it on the Windows Desktop as a minimized application.

`{EXEC "C:\COMMAND.COM /C DIR>TEST.TXT",1}` stores the current directory listing in the file TEST.TXT.

See also {INITIATE}.

{EXECUTE}

Format {EXECUTE *DDEChannel, Macro<, ResultLoc>*}

DDEChannel = channel ID number of the application to run a macro in
Macro = string containing the macro command(s) to run
Result = a cell indicating the result of the operation

Some Windows applications refer to macros as scripts.

With {EXECUTE} you can make other applications that support DDE run their macros. The macro to run is stored in the string *Macro*. That macro must be in the syntax the application normally uses (refer to the documentation for each application). For

example, {PGDN} works as a macro command for Quattro Pro, but the macro command for it in Excel is [VPAGE(1)]. You must open a channel of conversation with the other application using {INITIATE} (which determines the value of *DDEChannel*) before using {EXECUTE}.

The contents of *ResultLoc* depend on the application you are contacting. Most often, if a macro succeeds, *ResultLoc* contains 1. If a macro fails, {EXECUTE} results in an error.

See page 131 for more about running macros using DDE.

Examples The following example initiates a DDE conversation with ObjectVision and tells ObjectVision to open the file NOTES.OVD.

```
channel        15
command        [@APPOPEN("NOTES.OVD")]
result         0

_new_vision    {INITIATE "VISION","TASKLIST.OVD",channel}
               {EXECUTE channel,+command,result}
```

See {REQUEST} and {POKE} for more information on receiving data from and sending data to other applications using DDE.

{ExportGraphic}

Format {ExportGraphic *Filename*, <*GrayScale?*(0 | 1)>, <*Compression?*(0 | 1 | 2)>}

Filename = name of the graphic file to export
GrayScale? = whether to gray-scale: no (0), yes (1)
Compression? = type of .TIF file compression to use: none (0), PackBits (1), LZW (2)

{ExportGraphic} is the command equivalent for Draw | Export. It saves selected graphic objects to one of several file types with optional gray-scaling and compression. For details, see Chapter 10 of the *User's Guide*.

{FileClose...}

Command equivalent	Equivalent to ...
{FileClose <*DoSave?* (0 | 1)>}	File | Close
{FileCloseAll <*DoSave?* (0 | 1)>}	File | Close All

{FileClose} closes all views of the active notebook; {FileCloseAll} closes all open notebooks. The optional argument *DoSave?* indicates whether to display a save prompt before closing files with changes. Use 1, the default, to prompt for changes; 0 suppresses save prompts. If *DoSave?* is set to 1, {FileSave} and related command equivalents automatically prompt for backup if a file with the same name exists (see "{FileSave...}").

{FileCombine}

Format {FileCombine *Filename*, <*Blocks*>, Add | Subtract | Multiply | Divide | Copy}

Filename = name of the file to combine
Blocks = block or blocks within *Filename* to combine (optional)

{FileCombine} is the command equivalent for Tools | Combine. If you use the Copy option, it copies all or part of a notebook into the active notebook (starting at the selected cell). Omit *Blocks* to combine an entire file. Use Add, Subtract, Multiply, or Divide to perform mathematical operations; the incoming data operates on existing data. For details, see Chapter 15 of the *User's Guide*.

{FileExit}

Format {FileExit <*DoSave?* (0 | 1)>}

DoSave? = whether to display a save prompt for modified files: no (0), yes (1); 1 is the default

{FileExit} is the command equivalent for File | Exit. It closes Quattro Pro. The optional argument *DoSave?* indicates whether to display a save prompt before closing files with changes. Use 1, the default, to prompt for changes; 0 suppresses save prompts. If *DoSave?* is set to 1, {FileSave} and related command equivalents

automatically prompt for backup if a file with the same name already exists (see "{FileSave...}").

{FileExtract}

Format {FileExtract Formulas | Values, *Blocks, Filename*}

Blocks = block or blocks to extract
Filename = name of the new file containing *Blocks*

{FileExtract} is the command equivalent for Tools | Extract. It saves part of a notebook to a separate file, leaving the original file intact. Use Formulas to retain formulas; use Values to convert formulas to values. For details, see Chapter 15 of the *User's Guide*.

{FileImport}

{FileImport "ASCII Text File" | "Comma & " " Delimited File" | "Only Commas"} is the command equivalent for Tools | Import. It copies a text file into the active page of a notebook. Enter the option string that describes the type of file to import. For details, see Chapter 15 of the *User's Guide*.

{FileNew}

{FileNew} is the command equivalent for File | New. It opens a blank notebook. For details, see Chapter 5 of the *User's Guide*.

{FileOpen}

Format {FileOpen *Filename*}

Filename = name of the file to open

{FileOpen} is the command equivalent for File | Open. It opens the specified file. For details, see Chapter 5 of the *User's Guide*.

{FileRetrieve}

Format {FileRetrieve *Filename*}

Filename = name of the file to retrieve

{FileRetrieve} is the command equivalent for File | Retrieve. It loads a notebook into the active notebook, replacing any existing data there. For details, see Chapter 5 of the *User's Guide.*

{FileSave...}

Command equivalent	Equivalent to...
{FileSave <Replace | Backup | Confirm>}	File | Save
{FileSaveAll <Replace | Backup | Confirm>}	File | Save All
{FileSaveAs *Filename* <,Replace | Backup | Confirm>}	File | Save As

{FileSave} saves the active notebook, {FileSaveAll} saves all open notebooks, and {FileSaveAs} lets you save the active notebook under another name (*Filename*). The optional argument—Replace, Backup, or Confirm—indicates how to treat a previous version of the file (without displaying a prompt).

Examples To close all files and save without confirmation, use this macro:

```
{FileSaveAll Replace}
{FileCloseAll 0}
```

{FILESIZE}

Format {FILESIZE *Location*}

Location = any cell address or block name

{FILESIZE} calculates the number of bytes in a file previously opened using {OPEN} and places the result in *Location* as a value. If *Location* is a block, the result is placed in the upper left cell of the block. If the command is successful, macro execution continues in the cell below the cell containing the {FILESIZE} command, ignoring any other commands in that cell. If {FILESIZE} fails (for example, when no file is open), macro execution continues in the cell containing the {FILESIZE} command.

Examples The following example opens the file MYFILE.TXT and places its size (in bytes) in the cell named num_bytes.

```
\F            {OPEN "MYFILE.TXT",R}
              {FILESIZE num_bytes}
              {CLOSE}

num_bytes     45
```

{FLOATCREATE}

Format {FLOATCREATE *Type, UpperCell, xoffset, yoffset, LowerCell, xoffset2, yoffset2, Text*}

Type = floating object to create: Graph or Button
UpperCell = cell containing the upper left corner of the graph or macro button
xoffset = offset in twips from the left edge of *UpperCell* to the left edge of the floating object
yoffset = offset in twips from the top edge of *UpperCell* to the top edge of the floating object
LowerCell = cell containing the lower right corner of the graph or macro button
xoffset2 = offset in twips from the left edge of *LowerCell* to the right edge of the floating object
yoffset2 = offset in twips from the top edge of *LowerCell* to the bottom edge of the floating object
Text = for Graph, the named graph to display; for Button, the button text

{FLOATCREATE} lets you create a floating graph or macro button in the active notebook window. Use {CREATEOBJECT} to create objects in dialog windows or graph windows.

All positions in {FLOATCREATE} are positive offsets from cells in the notebook containing the upper left and lower right corners of the object.

If you need to modify the floating graph or button after creating it, change the property settings immediately after creation. It is selected then, so you won't need to click it or use {SELECTFLOAT}. You might also want to change the name at this time and document it for later use with {SELECTFLOAT} (see the first example).

See also {CREATEOBJECT}, {FLOATMOVE}, {FLOATSIZE}, and {SELECTFLOAT}.

Examples This macro creates a macro button that covers the block A1..B2, and stores the name of the button in A26:

```
{FLOATCREATE Button,A1,0,0,C3,0,0,"Save File"}
{GETPROPERTY A26,"Object_Name"}
```

Since the button name is stored in A26, we can use property commands to customize the button; for example, using the following macro after running the macro in the previous example renames the button `Backup Button` and assigns the macro {FileSave Backup} to it:

```
{SELECTFLOAT +A26}
{SETPROPERTY "Label_Text","Backup Button"}
{SETPROPERTY "Macro","{FileSave Backup}"}
```

The next macro creates a button 50 twips (a twip is 1/1440th of an inch) to the right and 50 twips below the upper left corner of the button in the previous example:

```
{FLOATCREATE Button,A1,50,50,C3,50,50,"Open File"}
```

This macro creates a floating graph (showing the graph named Graph3) that's offset 35 twips from the block C2..E10, but the same size:

```
{GraphNew Graph3}
{FLOATCREATE Graph,C2,35,35,E10,35,35,"Graph3"}
```

See also {CREATEOBJECT}.

{FLOATMOVE}

Format {FLOATMOVE *UpperCell, xoffset, yoffset*}

UpperCell = cell containing the new upper left corner of the floating object
xoffset = offset in twips from the left edge of *UpperCell* to the left edge of the floating object
yoffset = offset in twips from the top edge of *UpperCell* to the top edge of the floating object

{FLOATMOVE} lets you move a floating object in the active notebook window. The item to move is selected using

{SELECTFLOAT}. The new position in {FLOATMOVE} is specified as a positive offset from a cell in the notebook.

See also {MOVETO}, {FLOATSIZE}, and {SELECTFLOAT}.

{FloatOrder.*Option*}

Command equivalent	Equivalent to Block|Object Order...
{FloatOrder.ToBack}	... | Send To Back
{FloatOrder.Backward}	... | Send Backward
{FloatOrder.ToFront}	... | Bring To Front
{FloatOrder.Forward}	... | Bring Forward

{FloatOrder.*Option*} is the command equivalent for Block | Object Order. It works on selected objects to arrange layers of floating graphs and other floating objects. For details, see Chapter 11 of the *User's Guide*.

{FLOATSIZE}

Format {FLOATSIZE *UpperCell, xoffset, yoffset, LowerCell, xoffset2, yoffset2*}

UpperCell = cell containing the new upper left corner of the graph or macro button

xoffset = offset in twips from the left edge of *UpperCell* to the left edge of the floating object

yoffset = offset in twips from the top edge of *UpperCell* to the top edge of the floating object

LowerCell = cell containing the new lower right corner of the graph or macro button

xoffset2 = offset in twips from the left edge of *LowerCell* to the right edge of the floating object

yoffset2 = offset in twips from the top edge of *LowerCell* to the bottom edge of the floating object

{FLOATSIZE} lets you resize a floating object in the active notebook window. The item to resize is selected using {SELECTFLOAT}.

All positions in {FLOATSIZE} are positive offsets from a cell in the notebook.

See also {RESIZE}, {FLOATMOVE}, and {SELECTFLOAT}.

{FOR}

Format {FOR *CounterLoc,Start#,Stop#,Step#,StartLoc*}

CounterLoc = cell used to track the number of macro iterations
Start# = initial value to place in *CounterLoc*
Stop# = maximum value for *CounterLoc*
Step# = amount added to *CounterLoc* after each iteration
StartLoc = cell containing the subroutine to run

{FOR} repeatedly runs a macro subroutine beginning at *StartLoc*, creating a macro *loop*. Quattro Pro keeps track of how long the macro runs using *CounterLoc*. At the start of the loop, *CounterLoc* is set to *Start#*. Each time an iteration of the loop runs, *CounterLoc* is increased by *Step#*. When *CounterLoc* reaches or exceeds *Stop#*, execution stops.

Examples `{FOR D15,1,5,1,E30}` runs the subroutine beginning in cell E30 five times.

`{FOR D15,1,10,2,E30}` runs the subroutine five times because the counter increments by two during each iteration.

`{FOR D15,1,5,0,E30}` runs the subroutine continuously until you press *Ctrl+Break*, because adding 0 to the start value of 1 can never make the counter exceed 5.

`{FOR D15,1,0,1}` results in an error because no subroutine is specified.

The following macro creates "Qtr" labels for reports; the year displays above "Qtr1."

```
\Q        {; position cursor where labels should appear}
          {FOR counter, 1992,1995,1,_list}
          {; return to original position}
          {LEFT}{END}{LEFT}

counter

_list     {PUTCELL +counter}
          {BlockValues C(0)R(0), C(0)R(0)}
          {DOWN}
          {PUTCELL "Qtr1"}{RIGHT}
          {PUTCELL "Qtr2"}{RIGHT}
          {PUTCELL "Qtr3"}{RIGHT}
          {PUTCELL "Qtr4"}{RIGHT 2}{UP}
```

{FORBREAK}

Format {FORBREAK}

{FORBREAK} cancels the subroutine run by a {FOR} command and ends the processing of {FOR}. Macro execution continues normally with the command following the {FOR} command.

If {FORBREAK} appears anywhere other than in a subroutine called by {FOR}, macro execution ends, and Quattro Pro displays an error message.

{FORBREAK} is usually used in conjunction with {IF} to exit the macro if a specific condition is reached—for example, if an error condition is found.

Examples The following example shows {FORBREAK} terminating a loop used to enter names into a list. Enter STOP to stop the loop.

```
\F           {EditGoto list_top}
             {FOR counter,1,10000,1,_get_name}

_get_name    {GETLABEL "Enter next name: ",name_cell}
             {IF name_cell="STOP"}{FORBREAK}
             {LET @CELLPOINTER("address"),name_cell}{DOWN}

name_cell    STOP

counter      4

list_top
```

{Frequency.*Option*}

Command equivalent	Equivalent to Data|Frequency...
{Frequency.Bin_Block *Block*}	... | Bin Block
{Frequency.Go}	... | OK
{Frequency.Reset}	... | Reset
{Frequency.Value_Block *Block*}	... | Value Blocks

{Frequency.*Option*} is the command equivalent for Data | Frequency. It counts the number of cases in the value *Block* that fall within each interval specified in the bin *Block*. Use {Frequency.Bin_Block} and {Frequency.Value_Block}, then {Frequency.Go}. You can use {Frequency.Reset} before or after the

other commands to clear current settings. For details, see Chapter 14 of the *User's Guide.*

Examples The following macro counts the data in block C1..E13 of page A and groups it according to the intervals given in G1..G7; frequencies display in column H.

```
{Frequency.Value_Block A:C1..E13}
{Frequency.Bin_Block A:G1..G7}
{Frequency.Go}
```

{FUNCTIONS}

{FUNCTIONS} is equivalent to the Functions key, *Alt+F3*, which displays a menu of functions to type into the input line.

{GET}

Format {GET *Location*}

Location = cell in which to store the keystroke entered by the user

{GET} pauses macro execution, accepts one keystroke from the user (no carriage return is necessary), and stores the keystroke as a macro command in *Location*. It then continues macro execution. Any Quattro Pro keystroke can be recorded with {GET} except *Shift+F2, Pause, Caps Lock, Num Lock,* and *Scroll Lock*. Unlike { ? }, the keystroke isn't passed to Quattro Pro.

{GET} is useful for creating macros that run in the background while the user works. It can detect each key, and restrict the actions the user performs. You can use {LOOK} with {GET} to check the type-ahead buffer for user response. See also {MENUBRANCH}, {MENUCALL}, { ? }, {LOOK}, {GETNUMBER}, and {GETLABEL}.

Examples The following example asks if the user wants to continue.

```
\F          {RIGHT 10}
            {; display msg_area in top left of screen}
            {QGOTO}msg_area~
_again      Press Y to continue...~
            {GET keystroke}
            {IF keystroke<>"Y"}{BEEP 2}{HOME}{QUIT}
            {BEEP 4}{BRANCH _again}
```

```
keystroke
msg_area     Press Y to continue...
```

{GETDIRECTORYCONTENTS}

Format {GETDIRECTORYCONTENTS *Block*, *<Path>*}

Block = block to enter list of files into
Path = path and wildcard specifying the list (optional)

{GETDIRECTORYCONTENTS} enters an alphabetized list of file names (determined by the path and DOS wildcard specified by *Path*) into *Block*; if *Path* isn't included, {GETDIRECTORYCONTENTS} lists all the files in the current directory. *Path* must contain a DOS wildcard like *.BAT or *.*.

Caution! If *Block* is one cell, {GETDIRECTORYCONTENTS} will overwrite any information beneath the cell (if it finds more than one file). To restrict the file names to a specific block, set *Block* to a block larger than one cell.

Examples `{GETDIRECTORYCONTENTS A2,"C:*.*"}` fills column A (starting at row 2) with a list of the files in the root directory of drive C.

`{GETDIRECTORYCONTENTS A2..C7,"C:\QPW*.*"}` fills the block A2..C7 with a list of the files in the QPW directory on drive C. The first filename is stored in A2, the second in B2, and so on. If more than 18 files are found, the block is only filled with the first 18.

`{GETDIRECTORYCONTENTS C7,"C:\QPW\SAMPLES*.W??"}` fills column C (starting at row 7) with a list of the files in the QPW\SAMPLES directory on drive C that have file extensions beginning with W.

{GETLABEL}

Format {GETLABEL *Prompt,Location*}

Prompt = string displayed to the user as a prompt
Location = cell in which to store the user's response

{GETLABEL} pauses macro execution, displays *Prompt* in a dialog box, and accepts keystrokes until the user chooses OK or presses *Enter*. At that time, Quattro Pro stores the characters typed as a

label in *Location*. Unlike {GET}, it doesn't accept Quattro Pro's special keys (for example, *Home*).

Prompt can be a literal string, such as "Enter date:", or a text formula such as +B6 (which contains a label). The prompt string itself can be up to 70 characters long, and is truncated if longer. The response can be as long as 160 characters.

If {PANELOFF} is in effect, {GETLABEL} ignores it, letting the menus, status line, and input line be viewed and updated as the input is typed. Once input is completed (indicated by *Enter* or choosing OK), these portions of the display are turned off again.

See also {GET}, {GETNUMBER}, and {PANELOFF}.

Examples This example displays the label in B6 as the dialog box prompt, and then stores the result in cell D1:

```
{GETLABEL +B6,D1}
```

The following example stores the user's last name in the cell named last_cell:

```
\F          {GETLABEL "Enter your last name:",last_cell}
last_cell  Piercherd
```

{GETNUMBER}

Format {GETNUMBER *Prompt,Location*}

Prompt = string displayed to the user as a prompt
Location = cell in which to store the user's response

{GETNUMBER} is like {GETLABEL}, but accepts only a numeric value, a formula resulting in a numeric value, or the cell address or block name of a cell returning a value. If a text value is input, ERR is entered in the cell. The prompt can be up to 70 characters long, and the response can be as long as 160 characters.

Examples The following example stores the user's age in the cell named age_cell. If a number isn't entered, which the macro detects by checking to see whether age_cell contains ERR, it prompts again.

```
\F          {GETNUMBER "Enter your age:",age_cell}
            {IF @ISERR(age_cell)}{BRANCH \F}
age_cell   24
```

{GETOBJECTPROPERTY}

Format {GETOBJECTPROPERTY *Cell, Object.Property*}

Cell = cell in which to store the property setting
Object = name of the object to study
Property = property of the object to study

{GETOBJECTPROPERTY} lets you inspect objects in Quattro Pro without using the mouse, including objects normally not selectable (like the application title bar). Selectable objects, such as blocks and annotations, can also be studied with {GETPROPERTY}. See Appendix B for the syntax of *Object.Property*. Appendix B also contains a list of objects and properties you can use.

Examples `{GETOBJECTPROPERTY A23,"Active_Notebook.Zoom_Factor"}` stores the Zoom Factor property's current setting in cell A23.

`{GETOBJECTPROPERTY B42,"/File/Exit.Enabled"}` stores whether File | Exit is operational or not in the cell B42.

See also {SETPROPERTY} and {SETOBJECTPROPERTY}.

{GETPOS}

Format {GETPOS *Location*}

Location = cell in which to store the retrieved value

{GETPOS} places the position of the file pointer in *Location* as a value. If no file has been opened using {OPEN}, the command is ignored. The file pointer is a number corresponding to the position at which the next character written to the file will be placed. If the file pointer is 0, the next character written to the file with {WRITE} or {WRITELN} is placed at the beginning of the file. If the file already contains information, it's overwritten, beginning at the first position in the file. After the file is written to, the file pointer is positioned immediately after the last character written. If a file is newly created, then written to for the first time, the file pointer will be the size of the file.

If {GETPOS} succeeds, macro execution continues in the cell below the cell containing the {GETPOS} command, ignoring any commands in the same cell as {GETPOS}; if {GETPOS} fails, macro execution continues in the same cell.

Examples The following example opens the text file TEST.TXT, reads in a line, and calculates the length of the line. The line length is calculated by subtracting the starting position from the ending position, and then subtracting 1 from the result. This adjustment is necessary because the carriage return and linefeed characters (found at the end of each line in a typical text file) are stripped away by Quattro Pro when the text is read into cells, so the calculated length is one character longer than the actual length. The text file read here was created by the macro example provided in the {WRITELN} command description.

```
\F            {OPEN "A:TEST.TXT",R}
              {GETPOS start}
              {READLN input}
              {GETPOS end}
              {CLOSE}
              {LET num_char,+(end-start)-1}

input         This is a short line.
start         0
end           22
num_char      21
```

{GETPROPERTY}

Format {GETPROPERTY *Cell,Property*}

Cell = cell in which to store the property setting
Property = property of the selected object to study

{GETPROPERTY} lets you study the property settings of whatever object is selected. *Property* is the property to study (Appendix B contains a list of properties you can use); its setting is stored in *Cell*.

Examples `{GETPROPERTY A23,"Text_Color"}` stores the Text Color setting of the selected object in the cell A23.

`{GETPROPERTY B42,"Box_Type"}` stores the border type of the selected floating object in cell B42.

See also {SETPROPERTY} and {SETOBJECTPROPERTY}.

{GETWINDOWLIST}

Format {GETWINDOWLIST *Block*}

Block = block to store window names in

{GETWINDOWLIST} stores a list of the windows open on the Quattro Pro desktop in *Block*, including dialog windows and graph windows. Windows hidden by Window | Hide aren't included.

Caution! If *Block* is one cell, {GETWINDOWLIST} overwrites any information beneath the cell (if it finds more than one window open). To restrict the window names to a specific block, set *Block* to a block larger than one cell.

Examples `{GETWINDOWLIST A2..C5}` stores a list of open windows in the block A2..C5. The first window name is stored in A2, the second in B2, and so on. If more than twelve windows are open, only the first twelve are stored in the block.

See also {CHOOSE}.

{GOTO}

{GOTO} lets you move the selector to a specific location. You can also use {QGOTO} or the command equivalent {EditGoto}, but these commands select a block if one is specified; {GOTO} selects the upper left cell of that block. This macro command is included for compatibility with Quattro Pro for DOS.

{GRAPH}

{GRAPH} is equivalent to the Graph key, *F11*, which displays the current graph.

{GRAPHCHAR}

Format {GRAPHCHAR *Location*}

Location = cell address or the block name where the returned character should be stored

{GRAPHCHAR} returns the key a user presses to leave a graph or to remove a message box. The character is stored at *Location*. This command lets you create a graph or message that functions as a menu of choices. The character returned can be used to branch or display any other data.

{GRAPHCHAR} can be used with {MESSAGE} to record what key the user pressed to remove a message box. It captures only characters the user can normally type into a cell; special characters such as *Esc* aren't stored.

Examples The following example displays a message box (its contents being the_msg) and returns the character the user pressed to remove it. The key pressed is stored in a block named the_key.

```
\A          {MESSAGE the_msg,5,5,0}
            {GRAPHCHAR the_key}

the_key

the_msg     Press C to continue or
            any other key to quit...
```

The next example displays a graph and waits for the user to press a key. If the user presses *P*, *B*, or *A*, Quattro Pro displays a (predefined) pie graph, bar graph, or area graph, respectively. The process repeats until the user presses any key but those three.

```
the_graph    bar
the_key      b

_graphic     {GraphView}
_continue    {GRAPHCHAR the_key}
             {if @UPPER(the_key)="P"}{_draw "pie"}
             {if @UPPER(the_key)="B"}{_draw "bar"}
             {if @UPPER(the_key)="A"}{_draw "area"}

_draw        {DEFINE the_graph}
             {GraphView the_graph}
             {BRANCH _continue}
```

{GraphCopy}

Format {GraphCopy *FromGraph*, *DestGraph*, <*Style?*(0 | 1)>, <*Data?*(0 | 1)>, <*Annotations?*(0 | 1)>}

FromGraph = graph containing the style, data, or annotation objects to copy

DestGraph	= new graph (the copy)
Style?	= whether to copy properties that affect the appearance of the graph: yes (1), no (0)
Data?	= whether to copy graph data: yes (1), no (0)
Annotations?	= whether to copy annotation objects: yes (1), no (0)

{GraphCopy} is the command equivalent for Graph | Copy. It copies the style, data, and/or annotation objects from one graph to another within a notebook (but not between notebooks). For details, see Chapter 8 of the *User's Guide*.

{GraphDelete}

Format {GraphDelete *Name*}

Name = name of the graph to delete

{GraphDelete} is the command equivalent for Graph | Delete. It deletes the specified graph from the active notebook. For details, see Chapter 8 of the *User's Guide*.

{GraphEdit}

Format {GraphEdit *Name*}

Name = name of the graph to edit

{GraphEdit} is the command equivalent for Graph | Edit. It displays the specified graph in a graph window for editing. For details, see Chapter 8 of the *User's Guide*.

{GraphNew}

Format {GraphNew *Name*}

Name = name of the new graph

{GraphNew} is the command equivalent for Graph | New. It creates a new graph and displays it in a graph window. Any selected data is shown in the graph. For details, see Chapter 8 of the *User's Guide*.

{GRAPHPAGEGOTO}

{GRAPHPAGEGOTO} lets you move to the Graphs page of the active notebook (like pressing *Shift+F5* when a spreadsheet page is active). When the Graphs page is active, you can use {SELECTOBJECT} to select icons, and other object commands to manipulate them.

You can use {SELECTBLOCK} to move from the Graphs page to a spreadsheet page.

See also {GOTO} and {QGOTO}.

{GraphSettings.Titles}

Format {GraphSettings.Titles *Main, Sub, X-Axis, Y-Axis, Y2-Axis*}

Main = main title of the graph
Sub = title appearing below the main title of the graph
X-Axis = title of the graph's x-axis
Y-Axis = title of the graph's y-axis
Y2-Axis = title of the graph's secondary y-axis

{GraphSettings.Titles} is equivalent to Graph | Titles, which sets the titles of the active graph (or selected floating graph or selected graph icon). For details, see Chapter 9 of the *User's Guide*. Each argument is a string; to reset a title, use an empty string ("").

Examples The following macro command displays the graph Profit99 in a graph window and sets its main and sub titles. The empty strings ("") indicate that there are no axis titles.

```
{GraphEdit Profit99}
{GraphSettings.Titles "Projected Profits","1999","","",""}
```

{GraphSettings.Type}

Command equivalent	Equivalent to Graph|Type...
{GraphSettings.Type *Type<,Class>*}	
{GraphSettings.Type "2DHalf Bar,2-D"}	... | 3-D | 2-5-D Bar
{GraphSettings.Type "3D Area,3-D"}	... | 3-D | 3-D |
{GraphSettings.Type "3D Bar,3-D"}	... | 3-D | Bar
{GraphSettings.Type "3D Column,3-D"}	... | 3-D | Column
{GraphSettings.Type "3D Contour,3-D"}	... | 3-D | Contour

{GraphSettings.Type "3D Pie,3-D"} ... | 3-D | Pie
{GraphSettings.Type "3D Ribbon,3-D"} ... | 3-D | Ribbon
{GraphSettings.Type "3D ShadedSurface,3-D"} ... | 3-D | Shaded Surface
{GraphSettings.Type "3D Stacked Bar,3-D"} ... | 3-D | Stacked Bar
{GraphSettings.Type "3D Step,3-D"} ... | 3-D | Step
{GraphSettings.Type "3D Surface,3-D"} ... | 3-D | Surface
{GraphSettings.Type "3D Unstacked Area,3-D"} ... | 3-D | Unstacked Area
{GraphSettings.Type "Area,2-D"} ... | 2-D | Area
{GraphSettings.Type "Area_bar,Combo"} ... | Combo | Area-Bar
{GraphSettings.Type "Bar,2-D"} ... | 2-D | Bar
{GraphSettings.Type "Column,2-D"} ... | 2-D | Column
{GraphSettings.Type "HiLo,2-D"} ... | 2-D | High Low
{GraphSettings.Type "Hilo_bar,Combo"} ... | Combo | High Low-Bar
{GraphSettings.Type "Line,2-D"} ... | 2-D | Line
{GraphSettings.Type "Line_bar,Combo"} ... | Combo | Line-Bar
{GraphSettings.Type "Multiple 3D columns,Combo"} ... | Combo | 3D columns
{GraphSettings.Type "Multiple 3D Pies,Combo"} ... | Combo | 3D Pies
{GraphSettings.Type "Multiple Bar,Combo"} ... | Combo | Bar
{GraphSettings.Type "Multiple Columns,Combo"} ... | Combo | Columns
{GraphSettings.Type "Multiple Pies,Combo"} ... | Combo | Multiple Pies
{GraphSettings.Type "Pie,2-D"} ... | 2-D | Pie
{GraphSettings.Type "R2D Bar,Rotate"} ... | Rotate | 2D Bar
{GraphSettings.Type "R2DHalf Bar,Rotate"} ... | Rotate | 2.5D Bar
{GraphSettings.Type "R3D bar,Rotate"} ... | Rotate | 3D Bar
{GraphSettings.Type "Rotated Area,Rotate"} ... | Rotate | Area
{GraphSettings.Type "Rotated Line,Rotate"} ... | Rotate | Line
{GraphSettings.Type "Stacked Bar,2-D"} ... | 2-D | Stacked Bar
{GraphSettings.Type "Text,Text"} ... | Text
{GraphSettings.Type "Variance,2-D"} ... | 2-D | Variance
{GraphSettings.Type "XY,2-D"} ... | 2-D | XY

{GraphSettings.Type} is equivalent to Graph | Type, which lets you specify how the data in a graph is displayed. It affects the active graph (or graph icon or floating graph). For details, see Chapter 8 of the *User's Guide*. *Class* is an optional argument that specifies the class of graph type being used.

Examples `{GraphSettings.Type "Pie"}` and `{GraphSettings.Type "Pie,2-D"}` make the active graph a 2-D pie graph.

{GraphView}

Format {GraphView <*GraphName1, GraphName2,...*>}

GraphName1 = name of the first graph to display (optional)
GraphName2 = name of the second graph to display (optional)

{GraphView} is equivalent to Graph | View, which lets you display a graph (or series of graphs) full screen. For details, see Chapter 8 of the *User's Guide*. *GraphName* is the name of a graph to display. {GraphView} without an argument displays the active graph (or graph icon or floating graph).

Examples The following macro displays the named graphs Profit90 through Profit94.

```
{GraphView Profit90,Profit91,Profit92,Profit93,Profit94}
```

{GraphWindow.*Property*}

{GraphWindow.*Property*} is equivalent to Property | Graph Window (the graph window Object Inspector). For details, see Chapter 9 of the *User's Guide*. *Property* can be `Aspect_Ratio` or `Grid`.

{GraphWindow.Aspect_Ratio *Options*}

Sets the aspect ratio of the active graph. *Options* can be one of the following settings:

"35mm Slide"
"Floating Graph"
"Full Extent"
"Printer Preview"
"Screen Slide"

{GraphWindow.Grid *GridSize,DisplayGrid,SnapToGrid*}

Sets the grid size of the active graph window. Use *GridSize* to specify the percent of graph window between grid points; *DisplayGrid* specifies whether the grid is visible; *SnapToGrid* specifies whether the grid is active.

Examples The following macro sets up a 10 by 10 grid, displays the grid, and enables it.

```
{GraphWindow.Grid "10,Yes,Yes"}
```

{Group.*Option*}

Command equivalent	Equivalent to Tools|Define Group...
{Group.Define *GroupName, StartPage, EndPage*}	Tools | Define Group
{Group.Delete *GroupName*}	... | Delete
{Group.ResetNames}	Clears all group names in the notebook; no equivalent command

{Group.*Option*} is the command equivalent for Tools | Define Group. It creates and deletes page groups. For details, see Chapter 11 of the *User's Guide.*

Once you've defined a page group, you can use `{Notebook.Group_Mode "On"}` to activate Group mode. Use `"Off"` to cancel Group mode.

{GroupObjects}

{GroupObjects} is the command equivalent for Draw | Group. It groups selected objects in a graph window so they can be treated as one object in subsequent operations. Use {UngroupObjects} to treat them independently again. For details, see Chapter 10 of the *User's Guide.*

{HELP}

{HELP} displays a help screen. It is the equivalent of pressing *F1.*

{HLINE}

Format {HLINE *Distance*}

Distance = distance in columns to scroll the active notebook horizontally

{HLINE} scrolls the active notebook horizontally by *Distance* columns. If the number is positive, it scrolls right; if negative, it scrolls left. {HLINE} doesn't move the selector; only the view of the notebook is altered, just as if the scroll bars were used. Use the commands {RIGHT} or {LEFT} to move the selector horizontally.

Examples `{HLINE 10}` scrolls the display 10 columns to the right.

`{HLINE -5}` scrolls the display 5 columns to the left.

See also {LEFT}, {RIGHT}, {VLINE}, {VPAGE}, and {HPAGE}.

{HOME}

{HOME} is equivalent to the *Home* key.

{HPAGE}

Format {HPAGE *Distance*}

Distance = distance in screens to scroll the active notebook horizontally

{HPAGE} scrolls the active notebook horizontally by *Distance* screens. If the number is positive, it scrolls right; if negative, it scrolls left. {HPAGE} doesn't move the selector; only the view of the notebook is altered. Use {BIGRIGHT} or {BIGLEFT} to move the selector horizontally.

See also {BIGLEFT}, {BIGRIGHT}, {VLINE}, {VPAGE}, and {HLINE}.

{IF}

Format {IF *Condition*}

Condition = a logical expression (or the address of a cell containing a label, value, or expression)

{IF} operates like @IF. {IF} evaluates *Condition*; if it's numeric and nonzero, it's considered to be TRUE and macro execution continues in the same cell; if *Condition* is anything else (including 0), it's considered FALSE, and macro execution continues in the cell directly below the {IF} command. Unlike @IF, {IF} commands can't be nested within each other.

Examples The following example says, in English, "If the value stored in gpa is greater than 0.59, run the macro _pass_student, otherwise run the macro _fail_student."

```
\F      {IF gpa>.59}{BRANCH _pass_student}
        {BRANCH _fail_student}
```

If you don't want both your true and false clauses running, be sure to include a {BRANCH} or {QUIT} in the same cell as the {IF}, as shown in both examples. The following example uses a string of {IF} commands to award a grade based on a test result:

```
\G   {IF result>=.90}{BRANCH _give_a}
     {IF result>=.80}{BRANCH _give_b}
     {IF result>=.70}{BRANCH _give_c}
     {IF result>=.56}{BRANCH _give_d}
     {BRANCH _give_f}
```

{IFKEY}

Format {IFKEY *String*}

String = any macro name for a key (such as PGUP, END, GRAPH, and so on) without braces, or a string that returns a key macro name without braces

{IFKEY *String*} is like {IF}, but runs the next macro command in the current cell if *String* is the name of a key macro command (such as {ESC} or {HOME}). Don't enclose the name stored in *String* in braces. For example, {IFKEY "HOME"} evaluates as true because HOME is the name of a macro command; {IFKEY "A"} evaluates as false because it isn't. *String* can be a string or a text formula.

{ImportGraphic}

{ImportGraphic *Filename*}

Filename = name of the bitmap or other graphics file to import

{ImportGraphic} is the command equivalent for Draw | Import. It imports graphics files into a graph window. For details, see Chapter 10 of the *User's Guide*.

{INDICATE}

Format {INDICATE *String*}

String = any seven-character string

{INDICATE} sets the mode indicator in the lower right corner of the screen to read whatever is given as *String*. If *String* is longer than seven characters, only the first seven are used. To restore the mode indicator to its normal setting, use {INDICATE} with no arguments. To hide the mode indicator, use {INDICATE ""}.

Examples `{INDICATE "Save!"}` changes the indicator to read `Save!`.

`{INDICATE " Go! "}` changes the indicator to read `Go!` with a space preceding and following it.

`{INDICATE E14}` changes the indicator to `E14` because cell references are ignored.

`{INDICATE}` restores the normal mode indicator.

{INITIATE}

Format {INITIATE *AppName, Topic, ChannelLoc*}

AppName = name of the Windows application to communicate with
Topic = name of the file within the Windows application to communicate with
ChannelLoc = cell to contain the Channel ID number of the conversation if communication succeeds

{INITIATE} requests a channel of communication with the application *AppName*. This application must support Dynamic Data Exchange (DDE). If the request succeeds, the identification number of the conversation is stored in the cell *ChannelLoc*. *Topic* can be an Excel spreadsheet, an ObjectVision form, a Word for Windows document, or other DDE server-application file. If there is no specific file to communicate with in the other application, you can set *Topic* to `System`. *AppName*, *Topic*, or both can be an empty string. The empty string works as a wildcard to display a list of possible conversations to choose from (similar to the one in the next figure).

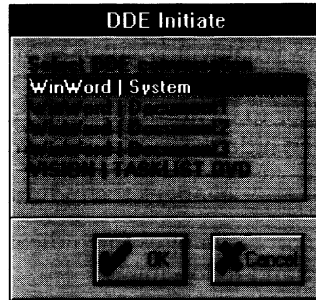


Figure 4.2
A DDE conversation list

You can use {INITIATE} multiple times with the same application; if you do this, store each instance of *DDEChannel* in a separate cell. Once a channel opens, you can use {REQUEST}, {EXECUTE}, or {POKE} to communicate with the application.

Communicating with another DDE application is called a *conversation*. Once your conversation is complete, use {TERMINATE} to end it. See {EXECUTE} for an example of {INITIATE}.

See also {EXECUTE}, {POKE}, and {REQUEST}. Page 131 tells more about using macros with DDE.

{INS}, {INSERT}, {INSOFF}, and {INSON}

{INS} and {INSERT} toggle the *Ins* key on or off. {INSOFF} is equivalent to *Ins* off, and {INSON} to *Ins* on.

{InsertBreak}

{InsertBreak} is the command equivalent for Block | Insert Break. It inserts a new line and a hard page break into notebook print blocks at the current selector location. For details, see Chapter 7 of the *User's Guide*.

{InsertObject}

Format {InsertObject *ObjectType*}

ObjectType = type of object to insert

{InsertObject} is equivalent to Edit | Insert Object, which lets you insert an OLE object into the active notebook without using the

Clipboard. For details, see Chapter 11 of the *User's Guide*. Use *ObjectType* to specify the type of object to insert.

Examples The following macro command inserts a picture created in Paintbrush into the active notebook.

```
{InsertObject "Paintbrush Picture"}
```

{Invert.*Option*}

Command equivalent	Equivalent to Tools|Advanced Math|Invert...
{Invert.Destination *Block*}	... | Destination
{Invert.Go}	... | OK
{Invert.Source *Block*}	... | Source

{Invert.*Option*} is the command equivalent for Tools | Advanced Math | Invert. It inverts a square matrix (indicated by {Invert.Source *Block*}) and stores the invert matrix in another block (indicated by {Invert.Destination *Block*}). Use {Invert.Go} after the other two matrix-inversion command equivalents to complete the operation.

You can use this command equivalent with {Multiply.*Option*} to solve sets of linear equations. For details, see Chapter 14 of the *User's Guide*.

{LEFT} and {L}

Format {LEFT <*Number*>}

Number = any positive integer (optional)

{LEFT} and {L} are equivalent to the ← key. The optional argument *Num* moves the selector the corresponding number of columns to the left. You can use cell references or block names as arguments.

Examples `{L}{L}{L}` moves the selector left three columns.

`{LEFT 6}` moves the selector six columns to the left.

`{LEFT D9}` moves to the left the number of columns specified in cell D9.

`{LEFT count}` moves to the left the number of columns specified in the first cell of the block named *count*.

{LET}

Format {LET *Location,Value<:Type>*}

Location = cell in which to store the specified value
Value = numeric or string value to be stored in *Location*
Type = `string` or `value`; `string` (or `s`) stores the value or formula as a label, and `value` (or `v`) stores the actual value or value resulting from a formula (optional)

With {LET}, you can enter a value into *Location* without moving to it. {LET} enters the value or string you specify with *Value* in *Location*.

You can use the optional *Type* argument to specify whether to store *Value* as an actual number or as a string. If you specify a formula as a string, the formula is written into *Location* as a string, not the resulting value. For example, {LET A1,B3*23:string} stores the formula B3*23 as a label in cell A1. If you omit the *Type* argument, Quattro Pro tries to store the value as a numeric value; if unsuccessful, it stores the value as a string.

Location must be a cell address or cell name; you can use functions such as @CELLPOINTER as a *Location* in {LET} commands only if they return a cell address or cell name.

You can use {LET} to invoke add-in @functions or macros contained in DLLs. Specify the add-in as *Value*, using this syntax:

@dllname.functionname(functionarguments)

The macro syntax is identical:

@dllname.macroname(macroarguments)

For example, this statement calls the @function MEDIAN, included in the DLL Stats, with a five-item list as an argument and stores the result in *Location* G6:

```
{LET G6,@Stats.MEDIAN(2,4,6,8,10)}
```

Examples `{LET (@CELLPOINTER("address")),99}` makes the value of the active cell 99.

The examples below assume A1 contains the label 'Dear, A2 contains the label 'Sir, and A3 contains the value 25. The result is shown to the right of each {LET}.

```
\M      {LET F1,25}                    25
        {LET F2,A3}                    25
```

```
{LET F3,+A1&" "&A2}             Dear Sir
{LET F4,+A1&" "&A2:value}       Dear Sir
{LET F5,+A1&" "&A2:string}      +A1&" "&A2
{LET F6,+A1&A3}                 ERR (because A3 is a
                                value)
```

{Links.*Option*}

Command equivalent	Equivalent to Tools...
{Links.Change *OldName, NewName*}	... | Update Links | Change Link
{Links.Delete *LinkName* | *} (* = all links)	... | Update Links | Delete Links
{Links.Open *LinkName* | *} (* = all links)	... | Update Links | Open Links
{Links.Refresh *LinkName* | *} (* = all links)	... | Update Links | Refresh Links

{Links.*Option*} is equivalent to commands on the Tools | Links menu, which let you refresh, change, or delete links in the active notebook. For details, see Chapter 12 of the *User's Guide*. *LinkName* is the name of the file being linked to. You can set *LinkName* to * to affect all links in the active notebook. If *LinkName* is omitted, the dialog box that normally performs the operation appears (and is under macro control; use {PAUSEMACRO} to pass control to the user).

Examples The following macro displays the Open Links dialog box and lets the user select the name of a linked notebook to open.

```
{Links.Open}
{PAUSEMACRO}
```

`{Link.Refresh *}` refreshes all links in the active notebook.

{LOOK}

Format {LOOK *Location*}

Location = a cell in which to store a typed character

When Quattro Pro runs a macro, it doesn't respond to keystrokes the user enters (except *Ctrl+Break*). If the user presses keys during macro execution, those keystrokes are stored in the computer's "type-ahead buffer" and are responded to when the macro pauses for input or ends.

{LOOK} checks this type-ahead buffer for stored keystrokes. If any are found, it places the first keystroke the user typed in *Location* as a macro command.

{LOOK} can be used while processing long macros to check for a keystroke that might signal the user wants to quit (see the example).

{LOOK} doesn't remove the keystroke from the buffer. If a macro does nothing other than {LOOK}, the key still passes to Quattro Pro when the macro ends. To remove pending keystrokes from the buffer, use {GET}.

Examples In the following example, the macro gives you 15 seconds to choose the next menu choice. If you don't, you must reenter the password.

Type the line to right of check_it with no hard returns until after {BRANCH _password}, then press Enter to insert it into one cell. We broke it up to print it on the page.

```
\M            {QGOTO}msg_area~
              A. Add name{DOWN}
              B. Edit name{DOWN}
              C. Delete name{DOWN}
              Enter choice: {RIGHT}
              { }
              {LET start_time,@NOW}
_check_it     {LOOK keystroke}{IF keystroke=""}{IF @NOW>start_time
              + @TIME(0,0,15)}{BRANCH _password}
              {IF keystroke<>""}{BRANCH _take_action}
              {BEEP 4}
              {BRANCH _check_it}

_password     {; get password from user}
              {GETLABEL "Enter password : ",pass}

_take_action  { }
              {BEEP 3}

start_time
keystroke
pass
msg_area
```

{MACROS}

{MACROS} is equivalent to the Macros key, *Shift+F3*, which displays a menu of macro commands to type into the input line.

{MARK}

{MARK} is equivalent to the Select key, *Shift+F7*, which lets you begin selection of a block.

{MENUBRANCH}

Format {MENUBRANCH *Location*}

Location = block containing a menu block

{MENUBRANCH} pauses macro execution to display the custom menu stored at *Location*. (Building a menu block is discussed next.) After the user chooses an item from the menu, macro execution continues with the cell below the description of the menu choice. Often this is a {BRANCH}. The only exception is if the user presses *Esc*, or clicks outside the menu; in this case, the macro continues in the {MENUBRANCH} cell.

With the exception of the *Esc* key, a custom menu acts the same as one of Quattro Pro's own menus. The user can use the arrow keys to look at each item, and can choose a menu item by either pressing *Enter* while highlighting it, pressing the first letter of the menu item's name, or choosing it with the mouse.

To build a menu block,

1. Locate an area of the notebook page with enough empty columns for each item in the menu, plus an extra blank column on the right.
2. Along the top row, place each menu item's title in a separate column, leaving no blank cells or values between items. These titles must be labels; values or blanks end the menu block.
3. In the row immediately below the titles row, place a label describing the menu item. The user will see this description on the status line when the item is highlighted on the menu. Values are ignored.
4. Place the desired macro commands for each of the menu items in the rows immediately below the descriptions.
5. To reference the menu by name instead of by address, give the upper left cell of the menu (the first menu item's title) a descriptive block name. It isn't necessary to name the entire contents of the custom menu because Quattro Pro continues to

add items to the menu until it reaches a blank column or value on the right.

Here are some suggestions for building custom menus:

- Never begin two or more items with the same letter. This prevents the user from choosing menu items by typing the first letter.
- Use the second line of the menus to fully describe the effect of choosing that item.
- Wherever appropriate, include a "Quit" or "Return to Quattro Pro" menu item as the last menu item. Forcing the user to press *Ctrl+Break* to return to Ready mode isn't "user friendly."
- The width of the columns containing the menu items has no bearing on the menu's appearance when it's displayed from a {MENUBRANCH}. However, choosing a remote area of the notebook and widening the columns that contain the titles, descriptions, and macros improves readability.

Examples The following macro displays a custom menu that offers the user three choices. If the user presses *Esc*, the menu is redisplayed with {BRANCH \F}.

```
quit_menu  Continue           Return            Quit
           Continue macro     Return to Ready   Leave Quattro Pro
           {BRANCH \F}        {BRANCH _cont}    {MENUBRANCH sure}

sure       No                 Yes
           Stay in macro      Return to DOS
           {BRANCH \F}        {FileExit}


\F         {MENUBRANCH quit_menu}
           {BRANCH \F}


_cont
```

See {ADDMENU} and {ADDMENUITEM} for information on adding menus or menu commands to the Quattro Pro menu bar.

{MENUCALL}

Format {MENUCALL *Location*}

Location = block containing a custom Quattro Pro menu

{MENUCALL} pauses macro execution and displays the custom menu stored at *Location*. It treats the called menu as a subroutine. After the user chooses an item from the menu, macro execution continues in the cell containing {MENUCALL}.

For a complete description of custom menus and how to build them, see {MENUBRANCH}.

Examples The following macro uses two consecutive custom menus to let the user change search criteria for a database. It then copies records that meet the criteria to the output block and branches to another macro to print the output block as labels. The block name stat references the cell in the criteria table containing A in the example; pay references the cell in the criteria table containing F.

```
m_status        Active                    Retired
                Active Members            Retired Members
                {LET stat, "A"}           {LET stat, "R"}


m_paid          Paid                      Unpaid
                Dues are paid             Dues are unpaid
                {LET pay, "T"}            {LET pay, "F"}


\M              {;Choose members for label print}
                {MENUCALL m_status}
                {MENUCALL m_paid}
                {Query.Extract}
                {BRANCH print_labels}
```

Database Criteria Block:

```
                STATUS                    PAID
                A                         F
```

See {ADDMENU} and {ADDMENUITEM} for information on adding menus or menu commands to the Quattro Pro menu bar.

{MESSAGE}

Format {MESSAGE *Block,Left,Top,Time*}

Block = block name or coordinates of the text and/or floating object to display in the message box
Left = screen column number (counting from 0) where the top left corner of the box should appear
Top = screen line number (counting from 0) where the top left corner of the box should appear
Time = Quattro Pro time serial number

{MESSAGE} displays the contents of *Block* in a dialog box until *Time* is reached; *Time* is a standard Quattro Pro date/time serial number (decimal portion). The window appears at the screen (not the notebook) coordinates *Left,Top*.

The message box contains a "snapshot" of current *Block* contents, including all fonts, colors, and other formatting. If *Block* includes a floating graph, bitmap, or macro button, its image displays in the message block.

The height and width of the message box depend on the defined width and depth of *Block*. If text overflows the cell it is typed into, be sure and include adjoining cells in *Block*. Otherwise, the message will be truncated at the edge of *Block*.

The message box always appears over the frontmost window—even if that window isn't a notebook (for example, a graph window).

If you enter `0` for *Time*, Quattro Pro displays the box until the user presses any key. (You can use {GRAPHCHAR} to test for which key is pressed.)

Examples This example displays a message box for 15 seconds and then displays another box indefinitely. The user removes the second box by pressing Y (which is returned in the block the_key).

```
the_key

\A          {MESSAGE msg1,15,5,+@NOW+@TIME(0,0,15)}
_warning    {MESSAGE msg2,15,5,0}
            {GRAPHCHAR the_key}
            {IF @UPPER(the_key)="Y"}{BRANCH _lose_data}
            {BRANCH _warning}

msg1        Warning! You may lose data if you continue.
```

```
msg2      Are you sure you want to continue?
          Press Y to continue or any other key to cancel...
```

Be sure to specify blocks wide enough to display each message without truncating text.

See also {DODIALOG}.

{MOVETO}

Format {MOVETO *x*, *y*}

x,*y* = position to move the currently selected object(s) to in pixels

{MOVETO} moves all selected objects in the active dialog window, graph window, or Graphs page to the position specified by *x*,*y*. Since {MOVETO} is context sensitive, you can use it to move controls in a dialog window or drawings in a graph window. It also moves graph icons on the Graphs page. The coordinates *x* and *y* represent where to move the upper left corner of the object(s). Object size doesn't change.

See also {RESIZE} and {SELECTOBJECT}. Use {FLOATMOVE} to move floating objects in the notebook window.

{Multiply.*Option*}

Command equivalent	**Equivalent to Tools|Advanced Math| Multiply...**
{Multiply.Destination *Block*}	... | Destination
{Multiply.Go}	... | OK
{Multiply.Matrix_1 *Block*}	... | Matrix 1
{Multiply.Matrix_2 *Block*}	... | Matrix 2

{Multiply.*Option*} is the command equivalent for Tools | Advanced Math | Multiply. It multiplies one matrix ({Multiply.Matrix_1 *Block*}) by another ({Multiply.Matrix_2 *Block*}) and stores the product in another block ({Multiply.Destination *Block*}). Use {Multiply.Go} after the other matrix-multiplication command equivalents to complete the operation.

You can use this command equivalent with {Invert.*Option*} to solve sets of linear equations. For details, see Chapter 14 of the *User's Guide*.

Examples This macro multiplies block C2..D6 by block C18..G19 and stores the results in the block with upper-left cell F1.

```
{Multiply.Matrix_1 A:C2..D6}
{Multiply.Matrix_2 A:C18..G19}
{Multiply.Destination A:F1
{Multiply.Go}
```

{NAME}

{NAME} is equivalent to the Choices key, *F3,* which displays a list of block names in the current notebook.

{NamedStyle.*Option*}

Command equivalent	Equivalent to Edit|Define Style...
{NamedStyle.Alignment General | Left | Right | Center}	... | Included Properties | Alignment
{NamedStyle.Define "*StyleName, Align?* , *NumericFormat?, Protection?, Lines?, Shading?, Font?, TextColor?* "}	... | Define Style For
{NamedStyle.Delete *StyleName*}	... | Delete
{NamedStyle.Font "*FontName, PointSize, Bold, Italic, Underline, Strikeout*"}	... | Included Properties | Font
{NamedStyle.LineDrawing *LeftLine, TopLine, RightLine, BottomLine*}	... | Included Properties | Line Drawing
{NamedStyle.Numeric_Format *NumericFormat*}	... | Included Properties | Format
{NamedStyle.Protection Protected | Unprotected}	... | Included Properties | Protection
{NamedStyle.Shading *ForegroundColor, BackgroundColor, Pattern*}	... | Included Properties | Shading
{NamedStyle.Text_Color 0-15}	... | Included Properties | Text Color

{NamedStyle.*Option*} is equivalent to Edit | Define Style, which lets you create styles in the active notebook. For details, see Chapter 11.

These command equivalents don't take effect until the command {NamedStyle.Define} is used to create (or modify) a style. The arguments *Align?* through *TextColor?* each specify one property to

include in the style; use 1 to include the property, 0 to exclude the property.

{NamedStyle.Font} sets the new typeface and size of text in the cell. See page 404 for more information on specifying a font in a macro command.

{NamedStyle.Shading} sets the shading of the cell; *ForegroundColor* and *BackgroundColor* are numbers from 0 to 15; each specifies a color on the notebook palette to use; *Pattern* is a string (`Blend1` through `Blend7`).

Examples The following macro creates a new style named RedNote, which makes the active block red, and sets a new font.

```
{NamedStyle.Font "Courier,10,Yes,No,No,No"}
{NamedStyle.Text_Color "4"}
{NamedStyle.Define RedNote,0,0,0,0,0,1,1}
```

{NEXTPANE}

Format {NEXTPANE <*CellAtPointer?*>}

CellAtPointer? = specifies which cell should be active when the pane switches (0 or 1, optional)

{NEXTPANE} lets you switch between the panes of a notebook window previously split using Window | Panes. The optional argument *CellAtPointer?* specifies whether the active cell in the pane will be at the location of the selector (1) or its previous position (0). This command is equivalent to the Pane key, *F6*.

See also {ACTIVATE} and {PANE}.

{NEXTWIN}

{NEXTWIN} is equivalent to the Next Window key, *Ctrl+F6*.

{Notebook.*Property*}

{Notebook.*Property*} is equivalent to Property | Active Notebook (the notebook Object Inspector). For details, see Chapter 16 of the *User's Guide*. Each command affects the active notebook. The next table lists the possible settings for *Property*.

Property	Property|Active Notebook option
Display	Display
Group_Mode	none
Macro_Library	Macro Library
Palette	Palette
Password	none
Recalc_Settings	Recalc Settings
Zoom_Factor	Zoom Factor

{Notebook.Display<.*Option*>}	Equivalent to Property|Active Notebook...
{NoteBook.Display "*VertScroll, HorizScroll, Tabs*"}	... | Display
{NoteBook.Display.Show_HorizontalScroller Yes | No}	... | Display | Horizontal Scroll Bar
{NoteBook.Display.Show_Tabs Yes | No}	... | Display | Page Tabs
{NoteBook.Display.Show_VerticalScroller Yes | No}	... | Display | Vertical Scroll Bar

{Notebook.Display.*Option*} is equivalent to options of the notebook property Display.

Examples The following macro command hides the vertical and horizontal scroll bars of the active notebook, and reveals the page tabs.

```
{Notebook.Display "No,No,Yes"}
```

{NoteBook.Group_Mode On|Off}

Activates or deactivates group mode. Equivalent to using the Group button (by the page tabs).

{NoteBook.Macro_Library Yes|No}

Makes the active notebook a macro library (Yes).

{Notebook.Palette<.*Option*>}	Equivalent to Property|Active Notebook...
{Notebook.Palette *Color1,Color2,...,Color16*}	... | Palette
{NoteBook.Palette.Color_*n* "*RedValue,GreenValue,BlueValue*"}	... | Palette
{NoteBook.Palette.Color_*n*.Greed *GreenValue*}	... | Palette | Edit Color
{NoteBook.Palette.Color_*n*.Blue *BlueValue*}	... | Palette | Edit Color
{NoteBook.Palette.Color_*n*.Red *RedValue*}	... | Palette | Edit Color

{Notebook.Palette.*Option*} is equivalent to the notebook property Palette, which lets you set the colors of the active notebook. The arguments of {Notebook.Palette} (*Color1* through *Color16*) each have three parts, separated by commas: *RedValue*, *GreenValue*, and

BlueValue. Each part is a number from 0 to 255 (see page 402 for more information on specifying a color in a macro command). You also edit a part individually (see the second example).

Examples The following macro sets the third color on the notebook palette to violet (Red 255, Blue 255).

```
{Notebook.Palette.Color_3 "255,0,255"}
```

The following macro sets the amount of blue in the fifth color to 135.

```
{Notebook.Palette.Color_5.Blue "135"}
```

{NoteBook.Password *Password*}

Sets the password of the active notebook. The next save operation encrypts the file on disk.

{NoteBook.Recalc_Settings Automatic|Background|Manual,Column-wise|Row-wise|Natural, *Iterations*}

Sets the recalculation options of the active notebook.

{NoteBook.Zoom_Factor 25-200}

Sets the zoom factor of the active notebook.

{NUMOFF} and {NUMON}

{NUMOFF} and {NUMON} are equivalent to *Num Lock* off and *Num Lock* on, respectively.

{ONERROR}

Format {ONERROR *BranchLocation,<MessageLocation>, <ErrorLocation>*}

BranchLocation = first cell of the macro to run in case of an error
MessageLocation = cell in which to store any error message (optional)
ErrorLocation = cell in which to store the address of the cell containing the macro error (optional)

Normally, if an error occurs during macro execution, the macro ends and you see an error message. {ONERROR} tells Quattro Pro to branch to a different location (*BranchLocation*) if an error is encountered and to store the error message in *MessageLocation* for future reference. Quattro Pro stores the location of the error in *ErrorLocation*. *MessageLocation* and *ErrorLocation* are optional. If *MessageLocation* is omitted, no error message will be displayed or stored.

Only one {ONERROR} command can be in effect at a time, so each time it's called, the most recent {ONERROR} replaces the previous one. If an error occurs, the {ONERROR} state is "used up" and must be redeclared. It's best to declare {ONERROR} at the very beginning of your macro, or at least before any procedure that is likely to result in an error.

In general, {ONERROR} won't capture macro programming errors, such as an incorrect sequence of commands, or an attempt to call a nonexisting subroutine. Nor will it detect syntax errors within a macro itself, such as an error resulting from the incorrect use of a macro keyword.

Typical errors that {ONERROR} detects include

- disk errors, such as a disk drive door left open, a full disk, or failure to find a file of a given name
- attempts to copy to a protected cell when a page's Protection property is enabled
- attempts to insert a row that would push a nonblank cell off the bottom of the notebook

Examples The following macro demonstrates {ONERROR} by first deliberately causing an error (trying to insert a row that would push text off the bottom of the notebook), and then beeping when that error occurs.

```
\G          {ONERROR _on_err,err_msg,err_loc}
            {EditGoto A8192}
            {PUTCELL "This is the end"}
            {UP}
            {BlockInsert.Rows C(0)R(0), "Entire"}

_on_err     {BEEP 5}

err_msg
err_loc
```

{OLE.*Option*}

Command equivalent	Equivalent to...
{OLE.Change_Link *FileName*}	OLE Link settings | Change Link
{OLE.Change_To_Picture}	OLE Object settings | Change to Picture
{OLE.Edit}	OLE Link settings | Edit, OLE Object settings | Edit
{OLE.Play}	OLE Link settings | Play, OLE Object settings | Play
{OLE.Unlink}	OLE Link settings | Unlink
{OLE.Update}	OLE Link settings | Update Now
{OLE.Update_All_Objects}	Updates all OLE objects in active notebook; no equivalent command

{OLE.*Option*} is equivalent to commands in the Object Inspector menus of OLE objects. For details, see Chapter 11 of the *User's Guide*. Each command affects the selected OLE object, except {OLE.Update_All_Objects}, which refreshes all OLE objects in the active notebook. The type of OLE object determines what command equivalents affect it:

Table 4.15
OLE commands

OLE type	Commands
Embedded	{OLE.Change_To_Picture}, {OLE.Edit}, {OLE.Play}
Linked	{OLE.Edit}, {OLE.Play}, {OLE.Update}, {OLE.Change_Link}, {OLE.Unlink}

Examples The following macro selects an OLE object named Embedded1, lets the user edit the data (in the OLE server), and then converts the object into a picture (disabling the OLE link).

```
{SELECTFLOAT Embedded1}
{OLE.Edit}
{OLE.Change_To_Picture}
```

{OPEN}

Format {OPEN *Filename,AccessMode*}

Filename = file name
AccessMode = `R`, `M`, `W`, or `A`

{OPEN} establishes a connection to *Filename* so you can use other file-access macro commands ({READ}, {WRITE}, and so on) on it. There are four different access modes:

- R (Read-Only). Allows only reading from this file, ensuring no changes are made to it. {WRITE} and {WRITELN} can't be used with a file opened as Read-Only.
- M (Modify). Opens an existing file for modification. All reading and writing commands can be used with a file opened for modification.
- W (Write). Opens a new file with the name given in {OPEN}. If a file already exists with that name, the existing file is erased. All reading and writing commands can be used with a file opened for writing.
- A (Append). Opens an existing file for modification. Allows writing to the selected file only, and positions the file pointer at the end of the file.

Filename's full name and access path must be given, and the entire name must be in quotes. For instance, to open and modify the file DATA.TXT in the subdirectory called FILES on drive C, use the command {OPEN "C:\FILES\DATA.TXT",M}. If any part of the access path or file name is left out, the file might not be found.

Caution! Although {OPEN} can provide access to any type of file (including .WB1 spreadsheet files), Quattro Pro's file-access macro commands are designed to work only with plain text files. Using these commands with any other file type isn't recommended and can result in corruption of that file. Make a backup copy of your file before using {OPEN} on it.

Once {OPEN} runs successfully, Quattro Pro skips to the next row of the macro and continues running instructions there. {OPEN *Filename*,R}, {OPEN *Filename*,A}, and {OPEN *Filename*,M} fail if the file isn't found. {OPEN *Filename*,W} fails if the supplied access path or file name is invalid. If {OPEN} fails, Quattro Pro continues running commands in the same cell as the {OPEN} command. (See the third example below for an example.) You can use {ONERROR} to trap some errors that occur in {OPEN}.

Examples The following example opens the file named MYDATA.TXT for reading. If the file doesn't exist in the default data directory, the command fails.

```
{OPEN "MYDATA.TXT",R}
```

The next example creates the file MYDATA.TXT on drive A for writing. If there's already a file on drive A with that name, it's erased. This command will fail only if there's no disk in drive A.

```
{OPEN "A:MYDATA.TXT",W}
```

The last example demonstrates how to do error trapping in an {OPEN} macro. When you press *Ctrl+H,* Quattro Pro attempts to open the file C:\MYDIR\DATA.TXT. If that succeeds, the macro stops, since there are no more commands in the row below. If {OPEN} fails (meaning the file doesn't exist or the disk is unformatted), the adjacent {BRANCH} runs. The subroutine _try_again then attempts to create the file. If that succeeds, the macro restarts, since {OPEN} should succeed now that the file has been created. If _try_again fails, probably an invalid directory path was entered. Therefore, the adjacent {BRANCH} goes to _bad_dir, which displays a relevant error message with pertinent instructions.

```
\H          {OPEN "C:\MYDIR\DATA.TXT",M}{BRANCH _try_again}
            {CLOSE}

_try_again {OPEN "C:\MYDIR\DATA.TXT",W}{BRANCH _bad_dir}
            {CLOSE}
            {BRANCH \H}

_bad_dir    {EditGoto err_loc}{BEEP 4}
            {PUTCELL "The directory C:\MYDIR\ doesn't exist."}
            {DOWN}
            {PUTCELL "}Create it, then try again."}
            {DOWN}

err_loc
```

{Optimizer.*Option*}

Command equivalent	**Equivalent to Tools|Optimizer...**
{Optimizer.Add *Constraint#, Cell, Operator, Constant*}	... | Constraints | Add
{Optimizer.Answer_Reporting *Cell*}	... | Options | Reporting | Answer Report Block
{Optimizer.Change *Constraint#, Cell, Operator, Constant*}	... | Constraints | Change
{Optimizer.Delete *Constraint#*}	... | Constraints | Delete
{Optimizer.Derivatives Central | Forward}	... | Options | Derivatives
{Optimizer.Detail_Reporting *Cell*}	... | Options | Reporting | Detail Report Block
{Optimizer.Estimate Quadratic | Tangent}	... | Options | Estimates

{Optimizer.Linear 0 | 1}	... | Optimizer | Options | Assume Linear
{Optimizer.Load_Model}	... | Options | Load Model
{Optimizer.Max_Iters *Value*}	... | Options | Max Iterations
{Optimizer.Max_Time *Value*}	... | Options | Max Time
{Optimizer.Model_Cell *Cell*}	... | Options | Load Model,
{Optimizer.Precision *Value*}	... | Options | Precision
{Optimizer.Reset}	... | Reset
{Optimizer.Save_Model}	... | Options | Save Model
{Optimizer.Search Conjugate | Newton}	... | Options | Search
{Optimizer.Show_Iters 0 | 1}	... | Options | Show Iteration Results
{Optimizer.Solution_Cell *SolutionCell* }	... | Goal | Solution Cell
{Optimizer.Solution_Goal Max | Min | None | "Target Value"}	... | Goal
{Optimizer.Solve}	... | Solve
{Optimizer.Target_Value *Value*}	... | Target Value
{Optimizer.Variable_Cells *Cell(s)*}	... | Variable Cells

{Optimizer.*Option*} is the command equivalent for Tools | Optimizer. It performs goal-seeking calculations. It also solves sets of linear and nonlinear equations and inequalities. For details, see Chapter 3 of the *User's Guide*.

Constraint# refers to a constraint's order in the constraint list. *Constant* may be a value or a cell containing a value. The *Value* for Target_Value may also be a value or a cell. Use {Optimizer.Solve} after the other commands to calculate the solution.

To save an Optimizer model, use {Optimizer.Model_Cell *Cell*} {Optimizer.Save_Model}. To load a model, use {Optimizer.Model_Cell *Cell* }{Optimizer.Load_Model}

Examples The following macro sets up an Optimizer problem designed to maximize the formula in D6 by varying cells B8..B10. Seven constraints limit the solution. All options have been changed from their default settings. T2 and G13 are the upper-left cells of the report blocks.

```
{Optimizer.Solution_cell A:D6}
{Optimizer.Solution_goal Max}
{Optimizer.Value 0}
{Optimizer.Variable_cells A:B8..A:B10}
{Optimizer.Add 1,"A:D8..A:D8",<=,"1000"}
{Optimizer.Add 2,"A:B8..A:B8",>=,"100"}
{Optimizer.Add 3,"A:B9..A:B9",>=,"100"}
{Optimizer.Add 4,"A:B10..A:B10",>=,"100"}
{Optimizer.Add 5,"A:D8..A:D8",>=,"500"}
{Optimizer.Add 6,"A:D9..A:D9",<=,"900"}
```

```
{Optimizer.Add 7,"A:D10..A:D10",<=,"110000"}
{Optimizer.Max_Time 50}
{Optimizer.Max_Iters 300}
{Optimizer.Precision 5E-05}
{Optimizer.Linear 1}
{Optimizer.Show_Iters 1}
{Optimizer.Estimate Quadratic}
{Optimizer.Derivatives Central}
{Optimizer.Search Conjugate}
{Optimizer.Detail_Reporting A:T2..A:T2}
{Optimizer.Answer_Reporting A:G13..A:G13}
{Optimizer.Solve}
```

{Order.*Option*}

Command equivalent	Equivalent to ...
{Order.Backward}	Dialog | Order | Send Backward, Draw | Send Backward
{Order.Forward}	Dialog | Order | Bring Forward, Draw | Bring Forward
{Order.ToBack}	Dialog | Order | Send to Back, Draw | Send to Back
{Order.ToFront}	Dialog | Order | Bring to Front, Draw | Bring to Front

{Order.*Option*} is equivalent to Draw | Order and Dialog | Order, which reorder overlapping objects in a graph or dialog window. For details, see Chapter 10 of the *User's Guide* or Chapter 6. Each command affects selected objects in the active window.

See also {FloatOrder.*Option*}

{Page.*Property*}

{Page.*Property*} is equivalent to Property | Active Page (the page Object Inspector). For details, see Chapter 4 of the *User's Guide*. Each command affects the active page(s). The next table lists the possible settings for *Property*.

Property	Property|Active Page option
Borders	Borders
Conditional_Color	Conditional Color
Default Width	Default Width
Display_Zeros	Display Zeros
Grid_Lines	Grid Lines
Label_Alignment	Label Alignment
Line_Color	Line Color
Name	Name
Protection	Protection

{Page.Borders<.*Option*>}	Equivalent to Property|Active Page...
{Page.Borders "*Row, Column*"}	... | Borders
{Page.Borders.Column_Borders Yes | No}	... | Borders | Column ... | Borders
{Page.Borders.Row_Borders Yes | No}	... | Borders | Row Borders

{Page.Borders.*Option*} is equivalent to the page property Borders, which lets you hide the row and column borders of the active page.

Examples The following macro command hides the row border and leaves the column border.

```
{Page.Border "No,Yes"}
```

{Page.Conditional_Color<.*Option*>}	Equivalent to Property|Active Page...
{Page.Conditional_Color "*Enable, SmallVal, GreatVal, BelowColor, NormalColor, AboveColor, ERRColor*"}	... | Conditional Color
{Page.Conditional_Color.Above_Normal_Color 0-15}	... | Conditional Color | Above Normal Color
{Page.Conditional_Color.Below_Normal_Color 0-15}	... | Conditional Color | Below Normal Color
{Page.Conditional_Color.Enable Yes | No}	... | Conditional Color | Enable
{Page.Conditional_Color.ERR_Color 0-15}	... | Conditional Color | ERR Color
{Page.Conditional_Color.Greatest_Normal_Value *Value*}	... | Conditional Color | Greatest Normal Value
{Page.Conditional_Color.Normal_Color 0-15}	... | Conditional Color | Normal Color
{Page.Conditional_Color.Smallest_Normal_Value *Value*}	... | Conditional Color | Smallest Normal Value

{Page.Conditional_Color.*Option*} is equivalent to the page property Conditional Color, which makes cells change text color (based on the value in the cell). Each color specified in these commands is a number from 0 to 15, corresponding to which color of the notebook palette to use (1 through 16).

Examples The following macro makes negative values red, values greater than 10,000 green, ERR cells cyan, and positive values less than 10,000 black (assuming the default notebook palette is used).

```
{Page.Conditional_Color "Yes,0,10000,4,3,5,7"}
```

{Page.Default_Width *Width*}

Sets the default column width of the active page. *Width* is the new column width in twips (a twip is 1/1440th of an inch).

Examples The following macro makes the default column width a half inch (720 twips).

```
{Page.Default_Width "720"}
```

{Page.Display_Zeros Yes|No}

Specifies whether formulas returning zero are displayed (Yes) in the active page.

{Page.Grid_Lines<.*Option*>}

{Page.Grid_Lines<.*Option*>}	Equivalent to Property|Active Page...
{Page.Grid_Lines "*Horizontal,Vertical*"}	... | Grid Lines
{Page.Grid_Lines.Horizontal Yes | No}	... | Grid Lines | Horizontal
{Page.Grid_Lines.Vertical Yes | No}	... | Grid Lines | Vertical

{Page.Grid_Lines.*Option*} is equivalent to the page property Grid Lines, which lets you hide or reveal grid lines in the active notebook page.

Examples The following macro hides the horizontal grid lines of the active page.

```
{Page.Grid_Lines "No,Yes"}
```

{Page.Label_Alignment Left|Right|Center}

Sets the default label alignment for the active page.

{Page.Line_Color 0-15}

Sets the color of line drawings on the active page. Each value from 0 to 15 specifies a different color on the notebook palette.

{Page.Name *NewName*}

Sets the name of the active page.

{Page.Protection Disable|Enable}

Enables or disables cell protection on the active page.

{PANELOFF}

Format {PANELOFF}

{PANELOFF} disables normal display of menus and prompts during macro execution when Quattro Pro's Macro Suppress-Redraw property is set to None.

It can speed up execution for macros that use keystrokes to walk through menus, since it saves Quattro Pro the time normally needed to draw its menus on the screen. Its effect is canceled by Quattro Pro once the macro stops running, so you needn't worry about locking the user out of the menus. To cancel its effect during macro execution, use {PANELON}.

Note {PANELOFF} doesn't disable menus created by {MENUCALL} and {MENUBRANCH} or subroutine calls that use menus or dialog boxes.

Use this command with {WINDOWSOFF} to completely disable normal screen updating.

See also {PANELON}, {WINDOWSON}.

{PANELON}

Format {PANELON}

{PANELON} enables display of menus and prompts that have been disabled with {PANELOFF}. {PANELON} has no effect if used without an accompanying {PANELOFF}. Therefore, it can be used repeatedly with no ill effects.

Use this command with {WINDOWSON} to completely restore normal screen updating.

See also {PANELOFF} and {WINDOWSON}.

{Parse.*Option*}

Command equivalent	Equivalent to Data|Parse...
{Parse.Create}	... | Create
{Parse.EditLine *NewEditLine*}	... | Edit
{Parse.Go}	... | OK
{Parse.Input *Block*}	... | Input
{Parse.Output *Block*}	... | Output
{Parse.Reset}	... | Reset

{Parse.*Option*} is the command equivalent for Data | Parse. It breaks long text strings into data fields according to a format line. For details, see Chapter 15 of the *User's Guide*.

{Parse.Input} indicates the block to parse. {Parse.Output} indicates the block to hold parsed data. Use {Parse.Create} to build the format line. {Parse.EditLine} lets you specify a new format line. {Parse.Create} and {Parse.EditLine} act on the active block.

Use {Parse.Go} after setting up input and output blocks and creating a format line. {Parse | Reset} clears previous settings.

Examples The macro in the next figure selects a text block, builds a format line in the first line of the selected block, identifies the format line and text as the parse input block, specifies an output block, and performs the parse:

Figure 4.3
{Parse.*Option*} command statements with data

	A	B	C	D	E	F	G	H
5								
6	V>>>*L>>>>>>*L>>>******L>>>>>>>>>>>*L>					{SelectBlock A:A6..A:D7}		
7	1968	Davis,	John	Harrisburg,	PA	{Parse.Create}		
8	1972	Lee,	Elizabeth	New York,	NY	{Parse.Input_Block A:A6..A:D8}		
9						{Parse.Output_Block A:A11}		
10						{Parse.Go}		
11	1968	Davis,	John	Harrisburg	PA			
12	1972	Lee,	Elizabeth	New York,	NY			

{PasteFormat}

Format {PasteFormat *LinkType*}

LinkType = format to paste object as

{PasteFormat} is equivalent to Edit | Paste Format, which lets you create OLE links to other applications and paste data into the notebook that's in a specific format. For details, see Chapter 11 of the *User's Guide*. Use *LinkType* to specify the paste format.

See also {EditPaste}.

Examples The following macro command pastes the data in the Clipboard as a bitmap into the active notebook.

```
{PasteFormat Bitmap}
```

{PasteLink}

{PasteLink} is equivalent to Edit | Paste Link, which lets you set up a DDE link to another application. For details, see Chapter 11 of the *User's Guide*.

See also {EditPaste}.

{PasteSpecial}

Format {PasteSpecial *Properties,Contents,Transpose,Blanks*}

Properties = `Properties` to paste properties from Clipboard; "" otherwise
Contents = `Formulas` to paste formulas from Clipboard; `Values` to paste formula results from Clipboard; "" otherwise
Transpose = `Transpose` to transpose data in the Clipboard and paste it; "" otherwise
Blanks = `NoBlanks` to omit blank cells when pasting; "" otherwise

{PasteSpecial} is equivalent to Edit | Paste Special, which lets you paste certain aspects of data in the Clipboard. For details, see Chapter 11 of the *User's Guide*.

Examples The following macro command pastes formula results from the Clipboard, and skips any blank cells in the Clipboard.

```
{PasteSpecial "",Values,"",NoBlanks}
```

See also {EditPaste}.

{PAUSEMACRO}

{PAUSEMACRO} is used with {DODIALOG} or a command equivalent invoked with ! (see page 129) to pause the macro so the user can "finish up" whatever dialog box is displaying. Once the user finishes using the dialog box (by choosing OK, or canceling it), macro execution resumes with any macro commands following the {PAUSEMACRO}.

Caution! Only use {PAUSEMACRO} when a dialog box is displaying, Quattro Pro is in FIND mode (using Data | Query), or Quattro Pro is in INPUT mode (using Data | Restrict Input). Otherwise, the macro pauses indefinitely.

Examples The following macro displays the Block | Copy dialog box, sets the From edit field to A1, activates the To edit field, and then waits for the user to complete the copy operation. Once the user finishes the dialog box, the macro beeps and moves down a cell.

```
_copy_a1        {BlockCopy!}
                {ALT+F}
                A1
                {ALT+T}
                {PAUSEMACRO}
                {BEEP}{DOWN}
```

See also {DODIALOG}.

{PGDN} and {PGUP}

Format {PGDN <*Number*>}
{PGUP <*Number*>}

Number = any positive integer or address of a cell containing a positive integer (optional)

{PGDN} and {PGUP} are equivalent to *PgDn* and *PgUp*, respectively. Use *Number* to specify how many times the operation is repeated; for example, {PGUP 7} is equivalent to pressing *PgUp* seven times.

See also {VPAGE} and {HPAGE}.

{POKE}

Format {POKE *DDEChannel, Destination, DataToSend*}

DDEChannel = channel ID number of the application to send information to
Destination = location in the application that receives the information being sent
DataToSend = block of cells containing the information to send to the application

{POKE} sends information to an application that supports Dynamic Data Exchange (DDE). This application is identified by *DDEChannel*. The type of application determines what *Destination* is; it could be a block of cells in Excel or a bookmark in Word for Windows. *DataToSend* is a block containing the information to send. You must use the command {INITIATE} to open a channel of conversation before you can use {POKE} (this also determines the value of *DDEChannel*).

See page 131 for more about using macros with DDE.

Examples The following example starts a conversation with TASKLIST.OVD, which is a file open in ObjectVision. It sets the ObjectVision field Task to the label stored in new_task, and unchecks the Completed check box. Then the new task is inserted into the task list.

```
dde_channel    10
command        [@INSERT("TASKS")]
exec_result    0
new_task       Call Jim re: task priorities
task_status    No

_new_vision    {INITIATE "VISION","TASKLIST.OVD",dde_channel}
               {POKE dde_channel,"Task",new_task}
               {POKE dde_channel,"Completed",task_status}
               {EXECUTE dde_channel,+command,exec_result}
```

See also {REQUEST}.

{Preview}

{Preview} is equivalent to File | Print Preview, which lets you preview a printout onscreen. For details, see Chapter 7 of the *User's Guide*. See also {Print.*Option*}.

{Print.*Option*}

{Print.*Option*} is equivalent to File | Named Settings, File | Page Setup, and File | Print. For details, see Chapter 7 of the *User's Guide*. The command equivalent {Print.PrintReset} resets print settings in all the dialog boxes displayed by these commands.

File|Named Settings command equivalent	Equivalent to File|Named Settings...
{Print.Create *NamedSetting*}	... | Create
	... | Update
{Print.Delete *NamedSetting*}	... | Delete
{Print.Use *NamedSetting*}	... | OK

These command equivalents are equivalent to File | Named Settings. *NamedSetting* is the named print setting to affect. To update an existing named setting, use {Print.Create}. {Print.Delete} removes a named setting from the active notebook. {Print.Use} sets the current print settings to those stored under the name.

File|Page Setup command equivalent	Equivalent to File|Page Setup...
{Print.Bottom_Margin *Value*}	... | Margins | Bottom
{Print.Center_Block Yes | No}	... | Options | Center Blocks
{Print.Footer *FooterString*}	... | Footer
{Print.Footer_Margin *Value*}	... | Margins | Footer
{Print.Header *HeaderString*}	... | Header
{Print.Header_Margin *Value*}	... | Margins | Header
{Print.Headers_Font "*Typeface, PointSize, Bold* (Yes | No), *Italic* (Yes | No), *Underline* (Yes | No), *Strikeout* (Yes | No)"}	... | Header Font
{Print.Left_Margin *Value*}	... | Margins | Left
{Print.Orientation Landscape | Portrait}	... | Print Orientation
{Print.Page_Breaks Yes | No}	... | Options | Break Pages
{Print.PageSetupReset}	... | Reset Defaults
{Print.Paper_Type *PaperSize*}	... | Paper Type
{Print.Print_To_Fit Yes | No}	... | Options | Print to Fit
{Print.Right_Margin *Value*}	... | Margins | Right
{Print.Scaling 1-1000}	... | Scaling
{Print.Top_Margin *Value*}	... | Margins | Top

These command equivalents are equivalent to File | Page Setup. When specifying a margin, the default measurement system is used (set in the Windows Control Panel). To use a specific

measurement system, place `in` (for inches) or `cm` (for centimeters) after the new margin setting (see the example). The new setting is converted into the default measurement system.

Examples The following macro sets the top and bottom margins to three centimeters, specifies landscape orientation, and sets the paper size to Legal.

```
{Print.Top_Margin "3 cm"}
{Print.Bottom_Margin "3 cm"}
{Print.Orientation Landscape}
{Print.Paper_Type "Legal 8 1/2 x 14 inch"}
```

File|Print command equivalent	Equivalent to File|Print...
{Print.All_Pages Yes | No}	... | Print Pages | All Pages
{Print.Block *Block*}	... | Print Blocks
{Print.Copies *Value*}	... | Copies
{Print.DoPrint}	... | Print
{Print.DoPrintGraph}	... | Print (selected graph)
{Print.End_Page_Number *Value*}	... | Print Pages | To
{Print.Start_Page_Number *Value*}	... | Print Pages | From

These command equivalents are equivalent to File | Print (the commands available for File | Print | Options are discussed next). {Print.DoPrint} prints the active notebook (or active graph) using current print settings. {Print.DoPrintGraph} provides a quick way to print a graph. If a floating graph is selected, {Print.DoPrintGraph} prints the graph being shown; if a graph icon is selected, {Print.DoPrintGraph} prints the graph represented by that icon; if a graph window is active, {Print.DoPrintGraph} prints the graph shown.

Examples The following macro selects an icon on the Graphs page named Report3 and prints the graph it represents.

```
{GRAPHPAGEGOTO}
{SELECTOBJECT Report3}
{Print.DoPrintGraph}
```

The following macro prints pages seven through twelve of a document. The print block is A3..C234.

```
{Print.Block A3..C234}
{Print.All_Pages No}
{Print.Start_Page_Number 7}
{Print.End_Page_Number 12}
```

```
{Print.DoPrint}
```

File|Print|Options command equivalent	Equivalent to File|Print|Options...
{Print.Between_Block_Formatting "Lines" | "Page Advance"}	... | Print Between Blocks
{Print.Between_Page_Formatting "Lines" | "Page Advance"}	... | Print Between 3D Pages
{Print.Cell_Formulas Yes | No}	... | Print Options | Cell formulas
{Print.Left_Heading *Block*}	... | Left Heading
{Print.Lines_Between_Blocks *Value*}	... | Print Between Blocks | Lines
{Print.Lines_Between_Pages *Value*}	... | Print ... | Between 3D Pages | Lines
{Print.Print_Borders Yes | No}	... | Print Options | Row/Column Borders
{Print.Print_Gridlines Yes | No}	... | Print Options | Gridlines
{Print.PrintOptionsReset} {Print.PrintReset} resets all print settings	... | Reset Defaults
{Print.Top_Heading *Block*}	... | Top Heading

These command equivalents are equivalent to the Options command (in the File | Print dialog box). {Print.Between_Page_Formatting} and {Print.Lines_Between_Pages} control the amount of space left between notebook pages (if the print block spans multiple pages). {Print.Between_Block_Formatting} and {Print.Lines_Between_Blocks} control space between the subblocks that make up a noncontiguous print block.

Examples The following macro specifies that three lines should be printed between each notebook page (if the print block spans multiple pages), and that row and column borders should print.

```
{Print.Between_Page_Formatting "Lines"}
{Print.Lines_Between_Pages 3}
{Print.Print_Borders Yes}
```

{PrinterSetup}

Format {PrinterSetup *Printer,Port,PrintToFile,Filename,ReplaceOption*}

Printer	= printer to send print jobs to
Port	= port where the printer is connected
PrintToFile	= 1 to send all print jobs to a binary file; 0 otherwise
FileName	= name of the binary file to send print jobs to

ReplaceOption = specifies what to do when the binary file already exists

{PrinterSetup} is equivalent to File | Printer Setup, which lets you specify what printer Quattro Pro sends print jobs to. It also lets you create binary files. For details, see Chapter 7 of the *User's Guide*. To create a binary file, set *PrintToFile* to 1, *FileName* to the name of the binary file, and use *ReplaceOption* to tell Quattro Pro what to do if the binary file already exists. The following table lists the possible settings for *ReplaceOption*.

Table 4.16 *ReplaceOption* settings

Setting	Description
0	Cancels the operation. The file won't be overwritten.
1	Overwrites the file.
2	Makes a backup copy of the file before overwriting it.
3	Appends the new printing to the end of the existing file. This option is only available when printing plain text files.

Examples The following macro creates a binary file using current print settings. The binary file is named REPORT.PRN. If the binary file already exists, it's overwritten.

```
{PrinterSetup "Epson LQ-2500","LPT1:",1,"REPORT.PRN",1}
{Print.DoPrint}
{PrinterSetup "Epson LQ-2500","LPT1:",0,"",1}
```

{PUT}

Format {PUT *Location,Column#,Row#,Value<:Type>*}

Location = block within which *Value* will be stored, either as a value or label, as specified by *Type*

Column# = number of columns into the specified block to store *Value*

Row# = number of rows into the specified block to store *Value*

Value = string or numeric value

Type = `string` or `value`; `string` (or `s`) stores the value or formula as a label, and `value` (or `v`) stores the actual value or value resulting from a formula (optional)

{PUT}, like {LET}, copies a value to a particular cell. However, instead of placing the value directly in the specified cell, {PUT}

copies *Value* into the cell that's offset *Column#* columns and *Row#* rows into *Location.*

{PUT} processes *Value* the same way {LET} does, including the use of `:string` (or `:s`) and `:value` (or `:v`). If neither of these two optional arguments is supplied, {PUT} tries to store the value as a numeric value; if unsuccessful, it stores the value as a label.

The values for *Column#* and *Row#* can be any number between 0 and one less than the number of columns or rows within *Location*, respectively. A value of 0 implies the first column or row, 1 implies the second, and so on. If *Column#* or *Row#* exceed the number of columns or rows in the block, the macro stops. ({ONERROR} cannot trap this error.)

See also {LET}.

Examples Each of the following examples assumes cell A41 contains the value 25, the block named numbers has been defined as A44..B50, and data is a cell containing the value 295.

`{PUT numbers,1,4,A41:value}` copies the value 25 into the cell at the intersection of the second column and the fifth row of the block numbers (cell B48).

`{PUT numbers,1,5,A41:s}` copies the string "A41" into the cell at the 2nd column and the 6th row of the block numbers (cell B49).

`{PUT numbers,1,6,data}` copies the contents of the block data to the 2nd column and 7th row of numbers (cell B50). If there is no block named data, this example instead places a label ("data") into cell B50.

{PUTBLOCK}

Format {PUTBLOCK *Value*<,*Block*>}

String= entry to type
Block = block to type *String* in (optional)

{PUTBLOCK} lets you quickly enter the same value, label, or formula in multiple cells. *String* is the string to type into *Block*. If *Block* isn't specified, the currently selected block is used. *Block* can be noncontiguous; if so, be sure and enclose it in parentheses. If *Value* is a formula containing relative addresses, those addresses are adjusted automatically (see the third example).

See also {PUTCELL}, {PUT}, and {LET}.

Examples `{PUTBLOCK "Quarter 1",A..D:A1}` enters the label `Quarter 1` in cells A:A1 through D:A1.

`{PUTBLOCK 1990,A..D:B1}` enters the value 1990 in cells A:B1 through D:B1.

`{PUTBLOCK "+A1",C3..C12}` enters the formula +A1 in C3, +A2 in C4, and so on.

{PUTCELL}

Format {PUTCELL *Data*}

Data = string to type into the active cell

{PUTCELL} is an easy way to enter information into the active cell.

See also {LET}, {PUT}, {GETNUMBER}, and {GETLABEL}.

Examples `{PUTCELL "Peggy Danderhoff"}` stores `Peggy Danderhoff` as a label in the active cell.

`{PUTCELL 45067}` stores the number 45067 as a value in the active cell.

`{PUTCELL @SUM(A1..A27)}` stores the formula @SUM(A1..A27) in the active cell.

{QGOTO}

{QGOTO} is equivalent to the GoTo key, *F5*, which lets you specify the active cell or select a block. You can also use the command equivalent {EditGoto}.

{QUERY}

{QUERY} is equivalent to the Query key, *F7*, which repeats the last Data | Query operation performed.

{Query.*Option*}

Command equivalent	Equivalent to Data|Query...
{Query.Assign_Names}	... | Field Names
{Query.Criteria_Table *Block*}	... | Criteria Table
{Query.Database_Block *Block*}	... | Database Block
{Query.Delete}	... | Delete
{Query.Extract}	... | Extract
{Query.Locate}	... | Locate
{Query.Locate} enters FIND mode, {Query.EndLocate} returns the user to the dialog box	
{Query.Output_Block *Block*}	... | Output Block
{Query.Reset}	... | Reset
{Query.Unique}	... | Extract Unique

{Query.*Option*} is equivalent to Data | Query, which lets you set up a Quattro Pro database and search for records in that database. For details, see Chapter 13 of the *User's Guide*. {Query.Locate} enters FIND mode and stays under macro control until {PAUSEMACRO} is used or {Query.EndLocate}, which exits FIND mode.

Examples The following macro sets up a database block and criteria table (A2..G37 and H1..H2), searches for records using the criteria table, and copies the first record found to A50.

```
{Query.Database_Block A2..G37}
{Query.Criteria_Table H1..H2}
{Query.Locate}
{BlockCopy []C(0)R(0)..C(6)R(0),A50}
{Query.EndLocate}
```

The following macro expands on the database set up in the previous example. It sets up an output block at J2..P2, and copies any records found by the previous search there.

```
{Query.Output_Block A2..G37}
{Query.Extract}
```

{QUIT}

{QUIT} ends all macro execution, and returns control of Quattro Pro to the user.

Use {QUIT} in conjunction with {IF}, {LOOK}, {MENUBRANCH}, or {MENUCALL} to end a macro under user control.

Examples The following macro displays a menu that has a "Quit" option, which returns the user to Ready mode.

```
quit_menu     Continue                  Quit
              Keep going                Quit to Ready mode
              {BRANCH \G}               {QUIT}

\G            {MENUBRANCH quit_menu}
```

{READ}

Format {READ *#Bytes,Location*}

#Bytes = number of bytes of characters to read from a file
Location = cell in which to store the characters read

See {GETPOS} for a discussion of the file pointer.

{READ} reads *#Bytes* bytes of characters from a file previously opened using {OPEN} (starting at the current position of the file pointer), and stores them as a label in *Location*. The file is left unchanged, and the file pointer moves to the position following the last character read.

{READ} is similar to {READLN}, except for two differences. While {READLN} reads one line of characters (terminated by a carriage-return/linefeed pair), {READ} reads the precise number of characters specified. This lets you read, for example, fields within a record rather than an entire record. The second difference is that while {READLN} strips out the carriage-return/linefeed pair at the end of a line, {READ} manipulates these as if they were no different from other characters. If you use {READ} to read a file created by {WRITELN}, you'll see two graphics characters at the end of each string read. These are the carriage return and linefeed characters. {READ} is best used only in conjunction with {WRITE}, or when you know the text file structure in detail.

If {READ} succeeds, macro execution continues in the cell below the cell containing the {READ} command; if {READ} fails, macro execution continues in the same cell.

Examples The following example opens a text file containing a phone directory database, and reads a name and phone number from the third record. The macro that created this text file is shown in the description of {WRITE}:

```
\L            {OPEN "A:PHONEDIR.PRN",R}
              {SETPOS rec_length*(rec_number-1)}
              {READ name_length,name}
              {READ phone_length,phone}
              {CLOSE}

rec_number    3
rec_length    27
name_length   14
phone_length  12
name          Hall, Sue Ann
phone         617-555-5678
```

{READLN}

Format {READLN *Location*}

Location = cell in which to store the characters read

{READLN} is like {READ}, but instead of using a number of bytes to determine the amount of text to read, {READLN} reads forward from the current file pointer location up to and including the carriage-return/linefeed at the end of the line. Unlike {READ}, it doesn't read the carriage return/linefeed into the cell.

Use {READ} to read lines from a record-structured file, where the lines are of uniform length. {READLN} can read the contents of a file one row at a time, making formatting of the data easier than {READ}.

Like the other file-access macros, if {READLN} fails, the macro continues execution in the current cell. If it's successful, the macro skips to the row below, and execution continues there. {ONERROR} can be used to trap disk and file errors, such as a disk drive door being left open.

Examples The following macro opens the text file TEST.TXT, reads in a line, and calculates the length of the line. The line length is calculated by subtracting the starting position from the ending position, and then subtracting 1 from the result. This adjustment is necessary because the carriage-return and linefeed characters found at the end of each line in a typical text file are stripped away by Quattro Pro when the text is read into cells. The macro that created the file TEST.TXT is shown in the description of {WRITELN}.

```
\M          {OPEN "A:TEST.TXT",R}
            {GETPOS start}
            {READLN input}
            {GETPOS end}
            {CLOSE}
            {LET num_char,+(end-start)-1}

start       0
end         22
num_char    21
input       This is a short line.
```

See {GETPOS} for a discussion of the file pointer.

{RECALC}

Format {RECALC *Location*<,*Condition*><,*Iteration#*>}

Location = block to recalculate
Condition = condition to be met before recalculation is halted (optional)
Iteration# = maximum number of times to recalculate *Location* trying to meet *Condition* (optional)

{RECALC} causes Quattro Pro to recalculate a specified portion of the notebook in a row-by-row order. This is different from normal recalculation, where Quattro Pro recalculates the entire notebook in natural order; that is, before a formula calculates, each cell it references is recalculated first.

With the optional *Condition* argument, you can tell Quattro Pro to recalculate formulas in a block repeatedly until the specified *Condition* is met. You can also supply an *Iteration#* argument to specify the maximum number of times to recalculate formulas trying to satisfy *Condition*. To use *Iteration#*, the *Condition* argument must also be supplied.

{RECALC} is useful for rapid recalculation of specified parts of a notebook, particularly when the notebook is so large that global recalculations would significantly slow your work.

{RECALC} overrides the recalculation method specified for the notebook, enforcing a row-by-row recalculation. If all the formulas reference only cells above, or to the left in the same row, the notebook will be correctly calculated. If there are references to cells to the left and below, you must use {RECALCCOL}. If there are

references to cells below or to the right in the same row as your formula, you must use {CALC} to recalculate the entire notebook.

{RECALC} displays the results of recalculation.

Caution! If there are formulas within the block being recalculated that depend on formulas outside of the block, they might not evaluate correctly. Make sure *Location* encompasses all the cells referenced by formulas within the block.

See {RECALCCOL} for an example using {RECALC}.

{RECALCCOL}

Format {RECALCCOL *Location<,Condition>,<Iteration#>*}

Location = cell block to recalculate
Condition = condition to be met before recalculation is halted (optional)
Iteration# = maximum number of times to recalculate *Location* trying to meet *Condition* (optional)

{RECALCCOL} is similar to {RECALC}, except {RECALCCOL} performs a columnwise recalculation instead of a rowwise calculation.

Examples This example shows a block of formula entries all linked to the value in the top left cell (C5), followed by the results of calculating the block with the Calc key (in Natural order), with {RECALCCOL}, and with {RECALC}. Each shows different results.

The next figure shows a block containing formulas. The upper left cell of the block is C5, which contains 1. The results of these formulas (when recalculated in natural order, with 1 in C5) are also shown.

Figure 4.4
A block of formulas (left) and their results (right)

	C	D
5	1	+C8+1
6	+C5+1	+D5+1
7	+C6+1	+D6+1
8	+C7+1	+D7+1

	C	D
5	1	5
6	2	6
7	3	7
8	4	8

The following macro sets recalculation to manual, enters 100 in the upper left cell of the block, and then recalculates the first column of the block (the results are shown after the macro):

```
\B       {Notebook.Recalc_Settings "Manual,Natural,1"}
```

```
{EditGoto C5}
{PUTCELL 100}
{RECALCCOL C5..C8}
```

Figure 4.5
Using {RECALCCOL}

	C	D
5	100	5
6	101	6
7	102	7
8	103	8

Notice that column D hasn't been recalculated, since it wasn't included in the block specified in the {RECALCCOL} command. The next macro recalculates the first row of the previous block (the results are shown after the macro):

```
\C      {RECALC C5..D5}
```

Figure 4.6
Using {RECALC}

	C	D
5	100	104
6	101	6
7	102	7
8	103	8

Notice the cells D6 through D8 aren't recalculated, since they aren't part of the block specified in the {RECALC} command.

{Regression.*Option*}

Command equivalent	**Equivalent to Tools|Advanced Math|Regression...**
{Regression.Dependent *Block*}	... | Dependent
{Regression.Go}	... | OK
{Regression.Independent *Block*}	... | Independent
{Regression.Output *Block*}	... | Output
{Regression.Reset}	... | Reset
{Regression.Y_Intercept Compute | Zero}	... | Y Intercept

{Regression.*Option*} is the command equivalent for Tools | Advanced Math | Regression. It performs a regression analysis to show the relationship between a set of independent variables and a dependent variable. For details, see Chapter 14 of the *User's Guide*.

{Regression.Dependent} indicates the dependent-variable block. {Regression.Independent} defines the independent variables. In {Regression.Independent}, *Block* can be noncontiguous with one variable to a column. The dependent and independent blocks must all have the same number of rows.

{Regression.Output} indicates where to store the table of regression results. {Regression.Y_Intercept} specifies whether to compute the Y-intercept, or set it to zero. You can use {Regression.Reset} to clear all settings. Use {Regression.Go} after the other command equivalents to perform the regression analysis. If data changes within the independent or dependent data blocks, use {Regression.Go} again to calculate a new regression table.

Examples The following macro sets these data blocks: Independent, B2..D16; Dependent, F2..F16. The last statement performs the regression analysis and stores the results in the block with upper-left cell H2.

```
{Regression.Independent_Block A:B2..A:D16}
{Regression.Dependent_Block A:F2..A:F16}
{Regression.Output_Block A:H2}
{Regression.Go}
```

{REQUEST}

Format {REQUEST *DDEChannel, DataToReceive, DestBlock*}

DDEChannel	= DDE channel number of the application to receive data from
DataToReceive	= information to receive from the application
DestBlock	= block to store the data received into

{REQUEST} gets information specified by *DataToReceive* from applications that support Dynamic Data Exchange (DDE). This information is stored in *DestBlock*. *DataToReceive* is a string representing the location of the data to receive in the other application. In Quattro Pro, this could be a block such as A2..A7. In ObjectVision this could be a field or record. You must use {INITIATE} to open a channel of communication and obtain the value *DDEChannel* before using {REQUEST}.

If your conversation isn't within a specific topic (in other words, you opened the channel using the command

{INITIATE *AppName*,"System",*DDEChannel*}), you can use the following strings in *DataToReceive*, depending on the application:

Table 4.17 Arguments for *DataToReceive*

String	Purpose
"SysItems"	A listing of all strings you can use with *DataToReceive*. You can use this command first to view other choices offered by *AppName*.
"Topics"	A listing of all topics open. For example, a list of open documents under Word for Windows.
"Status"	The current status of the application. For example, READY in Excel or EDIT in Quattro Pro when a cell is being edited.
"Formats"	A list of all Clipboard formats supported by the application or DDE link.
"Selection"	A list of all items currently selected in the application. For example, in Excel cells A3..A47 could be selected.

See page 131 for more about using macros with DDE. See also {POKE}.

Examples The following example gets information from the fields Task and Completed in the ObjectVision application TASKLIST.OVD and stores the data in the active notebook.

```
dde_channel       10
command           [@NEXT("TASKS")]
exec_result       0
vision_task       Print out third quarter report
task_complete     Yes

_get_vision_task  {INITIATE "VISION","TASKLIST.OVD",dde_channel}
                  {REQUEST dde_channel,"Task",vision_task}
                  {REQUEST dde_channel,"Completed",task_complete}
                  {EXECUTE dde_channel,+command,exec_result}
```

{RESIZE}

Format {RESIZE *x*, *y*, *NewWidth*, *NewHeight*, <*VertFlip?*>, <*HorizFlip?*>}

x and *y* = *xy* coordinates of the new upper left corner, in pixels
NewWidth = the new width, in pixels, of the object or group
NewHeight = the new height, in pixels, of the object or group
VertFlip? = 1 if the object or group is flipped vertically from its previous position

HorizFlip? = 1 if the object or group is flipped horizontally from its previous position

{RESIZE} resizes all selected objects in the active dialog window or graph window. Since {RESIZE} is context sensitive, you can use it to change drawings in a graph window or controls in a dialog window.

See also {FLOATSIZE} and {MOVETO}.

{ResizeToSame}

{ResizeToSame} is equivalent to Dialog | Align | Resize to Same, which lets you resize selected objects in the dialog window to the same size as the first object selected. For details, see Chapter 6.

{RESTART}

{RESTART} changes the current subroutine to the starting routine (or the main routine) by removing all preceding For loops and subroutine calls.

{RESTART} is typically used for error handling. If an application is already nested to near the maximum number of levels and a severe error occurs that requires the macro to end, {RESTART} ensures that additional subroutine calls can be made. If you use the {RESTART} command often, you may want to use {BRANCH} to run subroutines.

{RestrictInput.*Option*}

Command equivalent	Equivalent to...
{RestrictInput.Enter *Block*}	Data | Restrict Input (enters Input mode)
{RestrictInput.Exit}	Any operation that ends INPUT mode

{RestrictInput.*Option*} is equivalent to Data | Restrict Input, which lets you confine selector movement to a specific block of unprotected cells. For details, see Chapter 13 of the *User's Guide*. For an example, see Chapter 3. {RestrictInput.Enter} enters INPUT

mode and stays under macro control until {PAUSEMACRO} is used or {RestrictInput.Exit}, which exits INPUT mode.

{RETURN}

{RETURN} ends the running subroutine and returns control to the macro that called it. If the macro running isn't a subroutine, execution stops.

A {RETURN} command at the end of a subroutine is optional, since a macro automatically returns from a subroutine when it reaches a blank cell or a cell containing a value. {RETURN} is usually used in conjunction with {IF} to return to the main macro if a certain condition is met.

See {Subroutine} for an example of using {RETURN}.

{RIGHT} and {R}

Format {RIGHT <*Number*>}

Number = any positive integer (optional)

{RIGHT} and {R} are equivalent to the → key. The optional argument *Num* moves the selector the corresponding number of columns to the right. You can also use cell references or block names as arguments.

Examples `{R 6}` moves six columns to the right.

`{RIGHT D9}` moves right the number of columns specified in cell D9.

`{RIGHT count}` moves right the number of columns specified in the first cell of the block named count.

{ROWCOLSHOW}

Format {ROWCOLSHOW *Block*, *Show?*, *Row* or *Col*, *FirstPane?*}

Block = block containing rows or columns to hide or show

Show? = 1 to reveal rows or columns; 0 to hide rows or columns

Row or *Col* = 1 to reveal or hide a row; 0 to reveal or hide a column
FirstPane? = 1 to affect rows or columns in left or top window pane; 0 to affect them in the right or bottom window pane

{ROWCOLSHOW} lets you hide or reveal rows and columns (it is equivalent to the block property Reveal/Hide). *Show?* specifies whether to reveal (1) or hide (0). *Row* or *Col* specifies whether to affect rows (1) or columns (0). *Block* contains the rows or columns to affect. *FirstPane?* is used when the active window is split into panes (using Window | Panes). To affect the columns or rows in the left or top pane, set *FirstPane?* to 1; to affect rows or columns in the right or bottom pane, set *FirstPane?* to 0.

Examples `{ROWCOLSHOW A:A..B,1,0,1}` reveals columns A and B on page A.

`{ROWCOLSHOW A:1..7,0,1,1}` hides rows 1 through 7 on page A.

`{ROWCOLSHOW A:1..7,1,1,0}` reveals rows 1 through 7 on page A. If the window is split, the rows are revealed in the right or bottom pane.

See also {COLUMNWIDTH} and {ROWHEIGHT}.

{ROWHEIGHT}

Format {ROWHEIGHT *Block, FirstPane?,Set/Reset, Size*}

Block = block containing rows to resize
FirstPane? = 1 to resize rows in left or top window pane; 0 to resize rows in right or bottom window pane
Set/Reset = 0 to set the row height; 1 to reset the row height
Size = new height (in twips) if setting size; not needed if resetting size

{ROWHEIGHT} provides two ways to change the height of a row or block of rows (it is equivalent to the block property Row Height). The rows to change are specified by *Block*. *FirstPane?* is used when the active window is split into panes (using Window | Panes). To resize the rows in the left or top pane, set *FirstPane?* to 1; to resize the rows in the right or bottom pane, set *FirstPane?* to 0.

The argument *Set/Resize* specifies how to change the height. To set a row height, use the following syntax:

{ROWHEIGHT *Block, FirstPane?*, 0, *NewSize*}

NewSize is the new row height, in twips (a twip is 1/1440th of an inch). The maximum height is ten inches (14,400 twips).

To reset a row to the default height (determined by font sizes in the row), use the following syntax:

{ROWHEIGHT *Block, FirstPane?*, 1}

Examples `{ROWHEIGHT A:1..A:2,1,0,1440}` sets the height of rows 1 and 2 (on page A) to one inch (1,440 twips).

`{ROWHEIGHT A:1..A:2,0,0,2160}` sets the height of rows 1 and 2 (on page A) to one and a half inches (2,160 twips). If the window is split, the top or left pane is affected.

`{ROWHEIGHT A:5,1,1}` resets the height of row 5 (on page A) to the default height.

See also {COLUMNWIDTH} and {ROWCOLSHOW}.

{SCROLLOFF} and {SCROLLON}

The {SCROLLOFF} and {SCROLLON} macros are equivalent to *Scroll Lock* off and *Scroll Lock* on, respectively.

{Search.*Option*}

Command equivalent	Equivalent to Edit|Search and Replace...
{Search.Block *Block*}	... | Blocks
{Search.Case Any | Exact}	... | Case Sensitive
{Search.Direction Column | Row}	... | Columns First
{Search.Find *String*}	... | Find
{Search.Look_In Condition | Formula | Value}	... | Look In
{Search.Match Part | Whole}	... | Match Whole
{Search.Next}	... | Next
{Search.Previous}	... | Previous
{Search.Replace}	... | Replace button

{Search.ReplaceAll}	... | Replace All
{Search.ReplaceBy *String*}	... | Replace edit field
{Search.Reset}	... | Reset

{Search.*Option*} is equivalent to Edit | Search and Replace, which lets you search for strings in the active page. For details, see Chapter 3 of the *User's Guide*. Use {Search.ReplaceBy} to specify the replacement string; {Search.Replace} replaces the string.

Examples The following macro searches the active page for 1993 in formulas and replaces it with 1994.

```
{Search.Reset}
{Search.Block ""}
{Search.Look_In Formula}
{Search.Match Part}
{Search.Find "1993"}
{Search.ReplaceBy "1994"}
{Search.ReplaceAll}
```

{SELECTBLOCK}

Format {SELECTBLOCK *Block*}

Block = coordinates of the block(s) to select

{SELECTBLOCK} lets you select a contiguous or noncontiguous block within the active notebook. Parts of a noncontiguous block must be enclosed in parentheses.

See also {SELECTFLOAT} and {SELECTOBJECT}.

Examples `{SELECTBLOCK A4..B23}` selects the block A4..B23 in the active notebook window.

`{SELECTBLOCK (A:A1..A:B12,B:B13..B:C34)}` selects the noncontiguous block A:A1..A:B12, B:B13..B:C34.

{SELECTFLOAT}

Format {SELECTFLOAT *ObjectID1<, ObjectID2, ...>*}

ObjectIDx = name of the notebook object(s) to select

With {SELECTFLOAT} you can select floating objects in the active notebook window using their names. (To find the name of an

object, inspect it and study its Object Name property.) Use {SELECTOBJECT} to select objects in a graph or dialog window.

See also {FLOATMOVE}, {FLOATSIZE}, {SELECTBLOCK} and {SELECTOBJECT}. Appendix B gives details on working with objects and properties.

Examples `{SELECTFLOAT Button1}` selects the macro button in the active notebook window with the object name `Button1`.

{SELECTOBJECT}

Format {SELECTOBJECT *ObjectID1*<, *ObjectID2*, ...>}

ObjectIDx = identification number or name of the object(s) to select

With {SELECTOBJECT} you can select objects in the active window using their ID numbers or names. (To find the ID number of an object, inspect it and study its Object ID property. Its name is stored in its Name property.) Since {SELECTOBJECT} is context sensitive, you can select controls in a dialog window, drawings in a graph window, or icons on a Graphs page.

Appendix B gives details on working with objects and properties.

See also {SELECTFLOAT}.

Examples `{SELECTOBJECT 2,5,7}` selects the objects in the active window with the IDs 2, 5, and 7.

{Series.*Option*}

Command equivalent	Equivalent to Graph|Series...
{Series.Data_Range *SeriesNumber* | "XAxisLabelSeries" | "LegendSeries", *Block* <,*CreateIfNotExist?* (0 | 1)>}	... | Number *or* X-Axis *or* Legend
{Series.Delete *SeriesNumber* <,*AndAllSeriesFollowing?*>}	... | Delete
{Series.Go}	... | OK
{Series.Insert *SeriesNumber, Block*}	... | Add
{Series.Label_Range *SeriesNumber, Block* <,*CreateIfNotExist?* (0 | 1)>}	[series Object Inspector] | Label Series
{Series.Legend *SeriesNumber, LegendText*}	[series Object Inspector] | Legend
{Series.Reverse_Series 1 | 0}	... | Reverse Series
{Series.Swap_Row_Col 1 | 0}	... | Row/Column Swap

{Series.*Option*} is equivalent to Graph | Series (and options in a series Object Inspector), which let you create or delete graph series. For details, see Chapters 8 and 9 of the *User's Guide*. When you manipulate a series using command equivalents, the changes aren't made until the command {Series.Go} is used. In all the commands, *SeriesNumber* is the number of the series to affect (1 for the first series, 2 for the second, and so on).

{Series.Data_Range} changes the values of an existing series. *Block* is the new block the series should take values from. If you're not sure whether the series exists, set *CreateIfNotExist?* to 1. Then the series will be created if it doesn't already exist. You can also use {Series.Data_Range} to set the x-axis series (set *SeriesNumber* to XAxisLabelSeries) or set the legend series (set *SeriesNumber* to LegendSeries).

{Series.Delete} removes an existing series. Set *AndAllSeriesFollowing?* to 1 if you also want to remove all series following *SeriesNumber*.

{Series.Insert} creates a new series. The series is inserted at the position specified by *SeriesNumber*. *Block* is the block containing the new series' data.

{Series.Label_Range} sets up the labels for each value in a series. *Block* is the block containing the labels. If you're not sure whether the series exists, set *CreateIfNotExist?* to 1. Then the series will be created if it doesn't already exist.

{Series.Legend} sets the legend text for a series (*LegendText* is the new text).

Examples

The following macro creates a graph named Profit99 with two series. The series values are in A:A1..A27 and A:C1..C27. The series labels are in A:B1..B27 and A:D1..D27. The x-axis is stored in A:E1..E27.

```
{GraphNew Profit99}
{GraphEdit Profit99}
{Series.Data_Range 1,A:A1..A27,1}
{Series.Data_Range 2,A:C1..C27,1}
{Series.Label_Range 1,A:B1..B27}
{Series.Label_Range 2,A:D1..D27}
{Series.Data_Range "XAxisLabelSeries",A:E1..E27}
{Series.Go}
```

The following macro inserts a new series between the two series in the last example.

```
{GraphEdit Profit99}
{Series.Insert 2,A:G1..G27}
{Series.Go}
```

{SETGRAPHATTR}

Format {SETGRAPHATTR *FillColor, BkgColor, FillStyle, BorderColor, BoxType*}

FillColor = new fill color of the selected object(s)
BkgColor = new background color of the selected object(s)
FillStyle = new fill style of the selected object(s)
BorderColor = new border color of the selected object(s)
BoxType = new border style of the selected object(s)

{SETGRAPHATTR} lets you quickly set the properties of all selected objects in the active graph window. If one of the arguments specified in the {SETGRAPHATTR} command isn't appropriate for an object, that argument is ignored.

Each color specified in {SETGRAPHATTR} (*FillColor, BkgColor,* and *BorderColor*) is in quotes, and specified in RGB format (see page 402 for more information on specifying a color in a macro command).

See page 404 for a list of strings you can use for *FillStyle;* it must be in quotes.

BoxType specifies the new border style for the object; see the Border Style property in Appendix B for a list of strings you can use for *BoxType.*

See also {SETPROPERTY}, {SETOBJECTPROPERTY}, and {SPEEDFORMAT}.

{SETMENUBAR}

Format {SETMENUBAR *SystemDefinition*}

SystemDefinition = block containing a menu system definition

{SETMENUBAR} lets you specify which menu system displays on the menu bar. *SystemDefinition* is a block containing the new menu system definition. See Chapter 7 for a complete discussion of menu system definitions. You can use {SETMENUBAR} without an argument to restore the default Quattro Pro for Windows menu system.

Examples `{SETMENUBAR "A3..C324"}` makes the tree defined in A3..C324 the active menu system.

See also {ADDMENU} and {ADDMENUITEM}.

{SETOBJECTPROPERTY}

Format {SETOBJECTPROPERTY *Object.Property,Value*}

Object = object to alter property of
Property = property to alter
Value = new property setting (or another instance of *Object.Property* to copy the new setting from)

{SETOBJECTPROPERTY} can change the property settings of many Quattro Pro objects. Selectable objects such as blocks and annotations can also be changed using {SETPROPERTY}. *Object.Property* is the name of an object and property to set. Some of the objects {SETOBJECTPROPERTY} can affect include:

Dialog controls. Use this syntax to specify a control in a dialog window to manipulate:

[Notebook]DialogName:ObjectID.Property

[*Notebook*] is optional (without it, the active notebook is used). For example, the following macro sets the Fill Color property of the control Rectangle1 in the dialog ColorPick to red:

```
{SETOBJECTPROPERTY "ColorPick:Rectangle1.Fill_Color",
   "255,0,0"}
```

Graph objects. Use the same syntax as dialog controls, but substitute the name of the graph in place of *DialogName*. For example, the following macro changes the size of a rectangle named ColorPick in the graph 1QTR92:

```
{SETOBJECTPROPERTY "1QTR92:ColorPick.Dimension", "0,0,25,25"}
```

Menu items. Use the syntax *MenuPath.Property*. See the description of {ADDMENU} for the syntax of *MenuPath*. Chapter 7 lists the properties available in a menu item. For example, the following macro disables the Data menu in the active menu system.

```
{SETOBJECTPROPERTY "/Data.Disabled", "Yes"}
```

For a complete list of the objects and properties you can manipulate with {SETOBJECTPROPERTY}, along with more detailed instructions on identifying objects, see Appendix B.

Value is the new setting for the property. You can also Substitute another instance of *Object.Property* for this argument to copy property settings between objects. For example, the following macro copies the text color of the active block to the text color of A23:

```
{SETOBJECTPROPERTY "A1.Text_Color",
   "Active_Block.Text_Color"}
```

See also {GETPROPERTY}, {GETOBJECTPROPERTY}, and {SETPROPERTY}.

{SETPOS}

Format {SETPOS *FilePosition*}

FilePosition = the number of bytes into a file to set the file pointer to

See {GETPOS} for a discussion of the file pointer.

{SETPOS} moves the file pointer of a file previously opened using {OPEN} to the value *FilePosition*. (See {GETPOS} for a discussion of the file pointer.) *FilePosition* refers to the offset, in number of bytes, where you want to position the file pointer. Therefore, the first position in the file is numbered 0, not 1.

If no file is open when {SETPOS} is encountered (or some other problem occurs), macro execution begins with the next command in the same cell as {SETPOS}. If {SETPOS} succeeds, the rest of that cell's commands are ignored, and execution in the next row of the macro.

See also {ONERROR}. For an example using {SETPOS}, see {READ}.

{SETPROPERTY}

Format {SETPROPERTY *Property, Setting*}

Property = string representing the property to change
Setting = string representing the setting to apply to the property

{SETPROPERTY} alters the properties of the active object (selected with {SELECTBLOCK}, {SELECTFLOAT}, or {SELECTOBJECT}). *Property* is

the name of the property to alter, and *Setting* is the change to apply to that property.

Appendix B contains a list of properties and settings you can use and tells more about working with objects and properties.

Examples `{SETPROPERTY "Text_Color", "3"}` sets the selected block's Text Color property to the fourth color on the notebook palette.

The following macro selects the macro button named PushButton1 and renames it Button1.

```
name_float    {SELECTFLOAT "PushButton1"}
              {SETPROPERTY "Object_Name","Button1"}
```

{SHIFT}

Format {SHIFT+*Key* <*Number*>}

Key = keyboard macro command (PGUP, DOWN, and so on)

Number = number of times to repeat the operation (optional)

{SHIFT} emulates holding down the *Shift* key while pressing a keystroke. It's handy for selecting blocks, portions of an edit field, or parts of a formula being edited.

See also {ALT} and {CTRL}.

Examples `{SHIFT+DOWN 5}` emulates pressing the *Shift* key while moving down five cells.

`{EDIT}{END}{SHIFT LEFT 3}` edits the active cell, and selects the last three characters of the cell contents.

{Slide.*Option*}

Command equivalent	Action
{Slide.Effect *Effect*}	Specifies the transition effect to use when displaying the next slide in a slide show; no equivalent command
{Slide.Goto *SlideName*}	Takes the active slide show directly to the slide *SlideName*; no equivalent command
{Slide.Next}	Advances the active slide show to the next slide; no equivalent command

{Slide.Previous}	Returns the active slide show to the previous slide; no equivalent command
{Slide.Run *SlideShowName*}	Graph | Slide Show
{Slide.Speed *Speed*}	Specifies the transition speed to use when displaying the next slide in a slide show; no equivalent command
{Slide.Time *Time*}	Specifies the time in seconds to display the next slide in a slide show; no equivalent command

{Slide.*Option*} lets you build, edit, and present graphic slide show sequences. *Effect*, *Speed*, and *Time* are the same options offered in the Light Table dialog box. {Slide.Effect}, {Slide.Speed}, {Slide.Time}, {Slide.Goto}, {Slide.Next}, and {Slide.Previous} can be in the spreadsheet macro which started the slide show, in a spreadsheet macro run from a graph button, or attached directly to a macro button or button in a dialog box. For details, see Chapter 10 of the *User's Guide*.

{SolveFor.*Option*}

Command equivalent	**Equivalent to Tools|Solve For...**
{SolveFor.Accuracy *Value*}	... | Accuracy
{SolveFor.Variable_Cell *Cell*}	... | Variable Cell
{SolveFor.Formula_Cell *Cell*}	... | Formula Cell
{SolveFor.Go}	... | OK
{SolveFor.Max_Iters *Value*}	... | Max Iterations
{SolveFor.Reset}	Clears all Solve For settings; no equivalent menu command
{SolveFor.Target_Value *Value*}	... | Target Value

{SolveFor.*Option*} is the command equivalent for Tools | Solve For. It solves goal-seeking problems with one variable. For details, see Chapter 14 of the *User's Guide*.

{SolveFor.Formula_Cell} indicates the location of the formula to evaluate. {SolveFor.Target_Value} is the goal to reach, either a number or a cell containing a number. {SolveFor.Variable_Cell} indicates the formula variable (a referenced cell) that can change to reach the target value.

{SolveFor.Max_Iters} and {SolveFor.Accuracy} control how many calculation passes to make and how closely the solution must

match the target value. Use {SolveFor.Go} after the other commands. {SolveFor.Reset} clears previous settings.

{Sort.*Option*}

Command equivalent	Equivalent to Data|Sort...
{Sort.Block *Block*}	... | Block
{Sort.Data "Labels First" | "Numbers First"}	... | Data
{Sort.Go}	... | OK
{Sort.Key_*1-5 Block*}	... | Column | 1st – 5th
{Sort.Labels "Character Code" | "Dictionary"}	... | Labels
{Sort.Order_*1-5* Ascending | Descending}	... | Ascending | 1st – 5th
{Sort.Reset}	... | Reset

{Sort.*Option*} is equivalent to Data | Sort, which sorts the entries in a block. For details, see Chapter 13 of the *User's Guide*. The sort isn't performed until {Sort.Go} is used.

Examples The following macro sorts the block A3..C35 using two sort keys (columns A and C). The sort is in ascending order, and values in a column are placed in a group before labels in the column. The labels are sorted in dictionary order.

```
{Sort.Reset}
{Sort.Block A3..C35}
{Sort.Key_1 A25}
{Sort.Order_1 Ascending}
{Sort.Key_2 C23}
{Sort.Order_2 Ascending}
{Sort.Data "Numbers First"}
{Sort.Labels Dictionary}
{Sort.Go}
```

{SPEEDFILL}

{SPEEDFILL} is equivalent to the SpeedFill button on the SpeedBar. It fills the selected block with sequential data, based on the entries in the upper-left portion of the block.

{SPEEDFORMAT}

Format {SPEEDFORMAT *FmtName, NumFmt? , Font?, Shading?,TextColor?, Align?,LineDraw?, AutoWidth?,ColHead?, ColTotal?,RowHead?, RowTotal?*}

FmtName	= name of the format to apply
NumFmt?	= 1 to apply the numeric format; 0 otherwise
Font?	= 1 to apply the font; 0 otherwise
Shading?	= 1 to apply the shading; 0 otherwise
TextColor?	= 1 to apply the text color; 0 otherwise
Align?	= 1 to apply the alignment; 0 otherwise
LineDraw?	= 1 to apply the line drawing; 0 otherwise
AutoWidth?	= 1 to automatically size the columns; 0 otherwise
ColHead?	= 1 to apply the column heading format; 0 otherwise
ColTotal?	= 1 to apply the column total format; 0 otherwise
RowHead?	= 1 to apply the row heading format; 0 otherwise
RowTotal?	= 1 to apply the row total format; 0 otherwise

{SPEEDFORMAT} applies the format *FmtName* to the selected block; it is equivalent to the SpeedFormat button on the SpeedBar. The arguments *NumFmt* through *RowTotal?* each specify a part of the format to apply; use 1 to apply the part or 0 to omit the part.

See also {SETGRAPHATTR}, {SETOBJECTPROPERTY}, {SETPROPERTY}.

{SPEEDSUM}

Format {SPEEDSUM *Block*}

{SPEEDSUM} is equivalent to selecting a block and choosing the SpeedSum button from the SpeedBar. *Block* includes rows and/or columns to sum, plus adjacent empty cells to hold the results.

{STEP}

{STEP} is equivalent to the Debug key, *Shift+F2*.

{STEPOFF}

{STEPOFF} exits Debug mode, which runs the macro in slow-motion for debugging. The macro then runs at normal speed.

See {STEPON} for more information.

{STEPON}

{STEPON} enters Debug mode, which runs the macro one step at a time for debugging. When Quattro Pro encounters a {STEPON} command, it pauses and waits for the user to press any key or click the mouse before it runs the next macro command and pauses again. Debug mode continues until {STEPOFF} is encountered or the macro ends.

{STEPON} is like using the Debug key (*Shift+F2*) to enter Debug mode, but you can embed it within a macro. The Debug key must be used before a macro begins running. See page 139 for more information on debugging macros.

Examples The following example uses {STEPON} and {STEPOFF} to debug the last routine of a macro. Notice that the empty pairs of braces, placed at the beginning and end of the routine, make it easy to add {STEPON} and {STEPOFF} commands when there is a problem, because you can delete them when you want to debug, and insert them when you don't.

```
_clean_up     {; Save notebook to disk and say goodbye}
              { }{STEPON}
              {FileSave}
              {BRANCH _quit_msg}

_quit_msg     { }{STEPOFF}
              {QUIT}
```

{Subroutine}

Format *{Subroutine <ArgumentList>}*

Subroutine = name of the subroutine being called (which can be a block name or a cell address)

ArgumentList = list of one or more arguments to be passed to the specified subroutine (optional)

{Subroutine} calls the subroutine of the specified name, passing along any arguments given in *ArgumentList*. (Those arguments are then defined within the subroutine using the {DEFINE} command—see {DEFINE} for more information.)

Any macro can be a subroutine. The macro commands in the subroutine run until {RETURN}, a blank cell, or a value is encountered. Then Quattro Pro returns control to the calling routine, continuing execution with the macro command in the next cell. If the subroutine ends with {QUIT}, both the subroutine and the calling macro(s) end.

See page 125 for more information on subroutines.

See also {BRANCH} and {DISPATCH}.

Examples The following example invokes a subroutine that forces the user to verify printing twice before it begins.

```
_print        {GETLABEL "Print this (Y/N)?",ans_cell}
              {IF @UPPER(ans_cell)="Y"}{_chk_twice}
              {HOME}
                 .
                 .
                 .

_chk_twice    {; Double-check the print request}
              {GETLABEL "Are you certain (Y/N)?",ans_cell}
              {IF @UPPER(ans_cell)="N"} {RETURN}
              {GETLABEL "Ready to print now (Y/N)?",ans_cell}
              {IF @UPPER(ans_cell)="Y"}{Print.DoPrint}
              {RETURN}

ans_cell      Y
```

{TAB}

Format {TAB <*Number*>}

Number = any positive integer or the address of a cell containing a positive integer (optional)

{TAB}, like {BIGRIGHT}, is equivalent to *Ctrl* → or *Tab*; it selects the leftmost cell of the screen that's to the right of the current one.

The optional argument *Number* specifies how many times to repeat the operation; for example, {TAB 2} is equivalent to pressing *Tab* twice.

{TABLE}

{TABLE} is equivalent to the Table key, *F8*, which repeats the last Tools | What-If operation.

{TableQuery.*Option*}

Command equivalent	Equivalent to Data|Table Query...
{TableQuery.Destination *Block*}	... | Destination
{TableQuery.FileQuery Yes | No}	... | Query in File
{TableQuery.Go}	... | OK
{TableQuery.QueryBlock *Block*}	... | Query in Block (checked)
{TableQuery.QueryFile *Filename*}	... | QBE File

{TableQuery.*Option*} is equivalent to Data | Table Query, which lets you search external databases for records. For details, see *Database Access*. The query isn't performed until {TableQuery.Go} is used.

See also {Query.*Option*}.

Examples The following macro searches the external table TASKLIST.DB using the query file TASKLIST.QBE. The results of the search are stored in A:A2.

```
{TableQuery.FileQuery Yes}
{TableQuery.QueryFile TASKLIST.QBE}
{TableQuery.Destination A:A2}
{TableQuery.Go}
```

The next macro searches the same database, but uses the query defined in the named block task_query.

```
{TableQuery.FileQuery No}
{TableQuery.QueryBlock task_query}
{TableQuery.Destination A:A2}
{TableQuery.Go}
```

{TableView}

{TableView} is equivalent to Data | Database Desktop, which launches the Database Desktop. For details, see *Database Access*.

{TERMINATE}

Format {TERMINATE *DDEChannel*}

DDEChannel = channel number of the DDE conversation to terminate

{TERMINATE} closes down a DDE conversation opened with {INITIATE}. See {EXECUTE} for an example of {TERMINATE}.

See also {INITIATE}.

{UNDO}

{UNDO} "takes back" the last command given and restores the previous state for most commands.

{UngroupObjects}

{UngroupObjects} is the command equivalent for Draw | Ungroup. It separates the selected group of graph annotation objects so each can be moved or modified without affecting the others. For details, see Chapter 10 of the *User's Guide*.

{UP} and {U}

Format {UP <*Number*>}

Number = any positive integer (optional)

{UP} and {U} are equivalent to the ↑ key. The optional argument *Num* moves the selector up the corresponding number of rows. You can use cell references or block names as arguments.

Examples `{UP 4}` moves the selector up four rows.
`{UP C13}` moves the selector up the number of rows specified in cell C13.
`{UP temp}` moves the selector up the number of rows specified in the first cell of the block named temp.

{VLINE}

Format {VLINE *Distance*}

Distance = number of rows to scroll the active notebook vertically

{VLINE} scrolls the active notebook vertically by *Distance* rows. If the number is positive, it scrolls down; if negative, it scrolls up. {VLINE} doesn't move the selector; only the view of the notebook is altered. Use {DOWN} or {UP} to move the selector vertically.

Examples `{VLINE 11}` scrolls the display 11 rows down.

`{VLINE -4}` scrolls the display 4 columns up.

See also {HLINE}, {VPAGE}, and {HPAGE}.

{VPAGE}

Format {VPAGE *Distance*}

Distance = number of screens to scroll the active notebook vertically

{VPAGE} scrolls the active notebook vertically by *Distance* screens. If the number is positive, it scrolls down; if negative, it scrolls up. {VPAGE} doesn't move the selector; only the view of the notebook is altered. Use the commands {PGDN} or {PGUP} to move the selector vertically.

See also {HLINE}, {HPAGE}, and {VLINE}.

{WAIT}

Format {WAIT *DateTimeNumber*}

DateTimeNumber = the date and time at which macro execution can resume

{WAIT} pauses macro execution until the time corresponding to *DateTimeNumber* arrives. *DateTimeNumber* is a standard date/time serial number (both date and time portions must be included). If the current date is already later than the date specified in *DateTimeNumber*, macro execution continues immediately.

While execution is suspended, the macro is inactive, a WAIT indicator displays on the status line, and normal notebook operation can be restored only by pressing *Ctrl+Break*.

Examples The following macro beeps, displays a "Go home" message, and suspends all execution until 8:00 am the next day.

```
{BEEP 3}
{PUTCELL "Time to go home!"}
{DOWN}
{WAIT @TODAY+1+@TIMEVALUE("8:00 AM")}
```

The next macro waits for one day.

```
{WAIT @NOW+1}
```

This macro uses {WAIT} to alert the user by sounding five short beeps, spaced two seconds apart.

```
\H            {FOR counter,1,5,1,_loop_here}

_loop_here    {BEEP 3}
              {WAIT @NOW+@TIME(0,0,2)}

counter       6
```

{WhatIf.*Option*}

Command equivalent	Equivalent to Data|What-If...
{WhatIf.Block *Block*}	... | Data Table
{WhatIf.Input_Cell_1 *Cell*}	... | Input Cell (Column Input Cell)
{WhatIf.Input_Cell_2 *Cell*}	... | Row Input Cell
{WhatIf.One_Way}	... | One Free Variable, ... | Generate
{WhatIf.Reset}	... | Reset
{WhatIf.Two_Way}	... | Two Free Variables, ... | Generate

{WhatIf.*Option*} is the command equivalent for Data | What-If. It builds one- or two-variable "what-if" tables that display a range of results for different conditions. For details, see Chapter 14 of the *User's Guide*.

If you're creating a one-variable table, use these command equivalents: {WhatIf.Input_Cell_1}, {WhatIf.Block}, {WhatIf.One_Way}. For two-variable tables, use {WhatIf.Input_Cell_2} after indicating the first input cell; use {WhatIf.Two_Way} instead of {WhatIf.One_Way}.

Examples The following macro defines A4..H18 as the "what-if" block, B1 as Input Cell 1, B2 as Input Cell 2, and builds a two-variable table.

```
{Whatif.Block A:A4..A:H18}
{Whatif.Input_cell_1 A:B1}
{Whatif.Input_cell_2 A:B2}
{Whatif.Two_Way}
```

{WINDOW}

Format {WINDOW<*Number*>}

Number = number of an open notebook window (1-9)

{WINDOW<*Number*>} switches to the specified open notebook window. This command is for compatibility with the DOS version of Quattro Pro; use {ACTIVATE} to activate windows when developing macros for Quattro Pro for Windows. The argument *Number* is optional; {WINDOW} is equivalent to the Pane key, *F6*.

See also {ACTIVATE} and {CHOOSE}.

{WindowArrIcon}

{WindowArrIcon} is equivalent to Window | Arrange Icons, which lines up minimized windows on the Quattro Pro desktop. For details, see Chapter 6 of the *User's Guide*.

{WindowCascade}

{WindowCascade} is equivalent to Window | Cascade, which rearranges all open windows on the Quattro Pro desktop. For details, see Chapter 6 of the *User's Guide*.

{WindowClose}

{WindowClose} is equivalent to Close in a Control menu, which closes the active window (if the active window isn't saved, a prompt appears to confirm the operation).

See also {FileClose}.

{WindowHide}

{WindowHide} is equivalent to Window | Hide, which conceals the active notebook window. For details, see Chapter 6 of the *User's Guide*.

{WindowMaximize}

{WindowMaximize} is equivalent to Maximize in a Control menu, which enlarges the active window so it fills the screen.

{WindowMinimize}

{WindowMinimize} is equivalent to Minimize in a Control menu, which shrinks the active window to an icon on the Quattro Pro desktop.

{WindowMove}

Format {WindowMove *UpperLeftX*, *UpperLeftY*}

UpperLeftX = distance between the left side of the Quattro Pro window and the left side of active window, in pixels

UpperLeftY = distance between the bottom of the input line and the top of active window, in pixels

{WindowMove} is equivalent to Move in a Control menu, which lets you move the active window. *UpperLeftX* and *UpperLeftY* are the new coordinates of the upper-left corner of the window.

{WindowNewView}

{WindowNewView} is the command equivalent for Window | New View. It displays a duplicate copy of the active notebook in a new window. For details, see Chapter 6 of the *User's Guide*.

{WindowNext}

{WindowNext} is equivalent to choosing Next in a Control menu or pressing *Ctrl+F6*. It makes the next window active.

{WindowPanes}

Format {WindowPanes Horizontal | Vertical | Clear, *Synch?*(0 | 1), *Dim1*, *Dim2*}

Synch? = whether the panes are synchronized: yes (1) or no (0)
Dim1 = width of the left pane or height of the upper pane
Dim2 = width of the right pane or height of the lower pane

{WindowPanes} is the command equivalent for Window | Panes. It splits a notebook window into two horizontal or vertical panes; use Clear to restore a single pane. For details, see Chapter 6 of the *User's Guide*.

Dim1 and *Dim2* indicate the ratio relationship between the panes.

Examples This command equivalent splits the notebook window into two vertical panes, not synchronized. The first pane is twice as wide as the second:

```
{WindowPanes Vertical,0,2,1}
```

{WindowRestore}

{WindowRestore} is equivalent to Restore on the Control Menu. It restores minimized windows to their original size.

{WindowShow}

Format {WindowShow *Name*}

Name = name of the hidden window to show

{WindowShow} is the command equivalent for Window | Show. It shows hidden window *Name* and makes it active. For details, see Chapter 6 of the *User's Guide*.

{WindowSize}

{WindowSize *X*, *Y*}

X = new window width, in pixels
Y = new window height, in pixels

{WindowSize} is equivalent to Size in the Control menu. It sizes the active window to the specified width and height.

{WINDOWSOFF}

Format {WINDOWSOFF}

{WINDOWSOFF} disables normal screen updating during macro execution when Quattro Pro's Macro Suppress-Redraw property is set to None. It can speed up execution for most macros because it saves Quattro Pro the time normally needed to redraw the screen each time a cell changes. Quattro Pro cancels it once the macro stops running, so the user isn't "locked out" of the screen. To cancel its effect within the same macro, use {WINDOWSON}.

Use {WINDOWSOFF} along with {PANELOFF} to completely disable normal screen updating.

Examples The following macro uses {WINDOWSOFF} and {WINDOWSON} to turn off screen updating while Quattro Pro sorts a list of vendors with the block name vendor_name, thereby speeding up the sort operation.

```
sort_blk      vendor_name
key_nm        vendor_name

\W            {QGOTO}sort_message~
              {WINDOWSOFF}
              {_sort vendor_name}
              {WINDOWSON}

_sort         {DEFINE sort_blk}
              {Sort.Block @@(sort_blk)}
              {BlockCopy sort_blk,key_nm}
              {Sort.Key_1 @@(key_nm)}
              {Sort.Order_1 "Ascending"}
              {Sort.Go}

sort_message  SORT IS IN PROGRESS
```

```
vendor_name     General Cement Co.
                Alveoli Mfg., Inc.
                Sandab Development
                Consolidated Dust
```

Caution! After a {WINDOWSOFF} command, avoid letting a user point to a block in response to an Edit command. The selector may be in a different cell than the "frozen" display indicates. If a user must point to a block, precede it with a {WINDOWSON} command.

{WINDOWSON}

Format {WINDOWSON}

{WINDOWSON} reenables normal screen updating during macro execution, canceling the effects of a previous {WINDOWSOFF}. However, the screen won't be updated until a {CALC} macro command is encountered or the macro ends. If {WINDOWSON} is called when screen updating is already in effect (the normal mode), the command is ignored.

See {WINDOWSOFF} for an example using {WINDOWSON}.

{WindowTile}

{WindowTile} is the command equivalent for Window | Tile. It displays all open windows without overlapping them.

{WindowTitles}

{WindowTitles Horizontal | Vertical | Both | Clear} is the command equivalent for Window | Locked Titles. It locks specific rows and/or columns of a spreadsheet page onscreen as titles. When you scroll, the titles remain fixed onscreen while the rows below (or columns to the right) scroll as usual. Horizontal locks rows above the active cell, Vertical locks columns to the left of the active cell, and Both locks both rows and columns. Use Clear to "unlock" the titles. For details, see Chapter 6 of the *User's Guide*.

{WorkSpace.Option}

Command equivalent	Equivalent to File|Workspace...
{Workspace.Restore *Filename*}	... | Restore
{Workspace.Save *Filename*}	... | Save

{Workspace.Save} saves all open notebooks as a group with the specified *Filename* (Quattro Pro's default file extension for workspaces is .WSB). {Workspace.Restore} opens the specified file. For details, see Chapter 5 of the *User's Guide*.

{WRITE}

Format {WRITE *String<,String2,String3,...>*}

String = string of characters to be written into the open file

{WRITE} copies *String* to a file opened with the {OPEN} command, starting at the location of the file pointer. The file pointer is advanced to the position following the last character written, and the file's size increases if necessary.

{WRITE} doesn't place the carriage return and linefeed characters at the ends of lines. To include these characters, use {WRITELN}.

String can be a quoted string or text formula. If using more than one *String* argument, separate them with commas. For example, to write the label `Dear Ms.` followed by the contents of B6 into the file, use the following:

```
{WRITE "Dear Ms. ",+B6}
```

You can use an unlimited number of *String* arguments with {WRITE}.

If no file is open, or if there's insufficient room on the disk to increase the file's size, {WRITE} fails. Macro execution then continues with the first command after {WRITE} (in the same cell). If {WRITE} is successful, the rest of that cell's commands are ignored, and execution continues on the next row below.

If you try to reference a cell that doesn't contain a label, an error message displays.

Caution! After you finish accessing a file with {WRITE}, it *must* be closed with {CLOSE}. Otherwise, there's a good chance the file will

become corrupted if the computer is turned off or otherwise interrupted.

See also {WRITELN}.

Examples The following example creates a text file with fixed-length fields that will serve as a phone list database. After the macro runs, the file will look like this:

```
Golden, David 415-555-7774;Hack, Edmund  201-555-3511;Hall, Sue Ann
617-555-5678
```

See the description of {READ} to see how to read a file like this:

```
\K    {OPEN "A:PHONEDIR.PRN",W}
      {WRITE "Golden, David "}
      {WRITE "415-555-7774;"}
      {WRITE "Hack, Edmund  "}
      {WRITE "201-555-3511;"}
      {WRITE "Hall, Sue Ann "}
      {WRITE "617-555-5678"}
      {CLOSE}
```

See {GETPOS} for a discussion of the file pointer.

{WRITELN}

Format {WRITELN *String<,String2,String3,...>*}

String = string of characters to be written into the open file as a single line

{WRITELN} copies *String(s)* to a file opened with {OPEN}, starting at the location of the file pointer, and ends the string(s) with the carriage-return and linefeed characters. The file pointer advances to the position following the last character written, and the file's size increases if necessary.

To enter a blank line in the file, use {WRITELN ""}. To enter characters without carriage-return/linefeed characters, use {WRITE}.

String can be a quoted string or a text formula. If using more than one *String* argument, separate them with commas. For example, to write the label `Dear Ms.` followed by the contents of B6 into the file you'd use

```
{WRITELN "Dear Ms. ",+B6}
```

A carriage return and linefeed character is placed at the end of all strings combined, not after each. You can use an unlimited number of *String* arguments with {WRITELN}.

If no file is open, or if there's insufficient room on the disk to increase the file's size, {WRITELN} fails. Macro execution then continues with the first command after {WRITELN} in the same cell. If {WRITELN} succeeds, the rest of that cell's commands are ignored, and execution continues on the next row below.

Caution! After you finish accessing a file with {WRITELN}, you must close the file with {CLOSE}. Otherwise, there is a good chance the file will become corrupted if the computer is turned off or otherwise interrupted.

See also {WRITE}.

Examples The following example shows how {WRITELN} is used to write a line of text to the file TEST.TXT. This line is terminated by the CR (carriage-return) and LF (linefeed) characters.

```
\Z    {OPEN "A:TEST.TXT",W}
      {WRITELN "This is a short line."}
      {CLOSE}
```

{ZOOM}

{ZOOM} maximizes or restores the active window. This command is for compatibility with the DOS version of Quattro Pro; use {WindowMaximize} and {WindowRestore} when developing macros for Quattro Pro for Windows.

C H A P T E R

5

Application basics

As an application developer, you use your creativity and imagination to build custom applications. To help you get started as a developer, this chapter introduces:

- Quattro Pro applications
- a sample application
- techniques for designing and assembling application components
- techniques for planning and designing applications
- the Developer mode

The chapters that follow this one contain explicit instructions for creating application components and assembling them, with examples.

What's an application?

A Quattro Pro *application* is a combination of components that work together to make a task easier. The final look and feel of an application can be radically different from the standard Quattro Pro notebook.

When you build an application, you can create custom components, such as:

- dialog boxes

- SpeedBars
- menus

After you create these components, you assemble them into an integrated application notebook (such as the sample application discussed next). This application notebook contains the macros, dialog boxes, and menus that make up the application.

The developer vs. the user

Notice that this chapter addresses "you" as the developer:

- **Developer** refers to the person who is building an application and who is interested in creating an application.
- **User** refers to the person who runs a completed application and who is interested in using the application to perform tasks.

Looking at a sample application

To get a sense of how components work within a Quattro Pro application, open BUDGTRAK.WB1, a sample application in your Quattro Pro program directory. This displays an opening screen similar to the next figure:

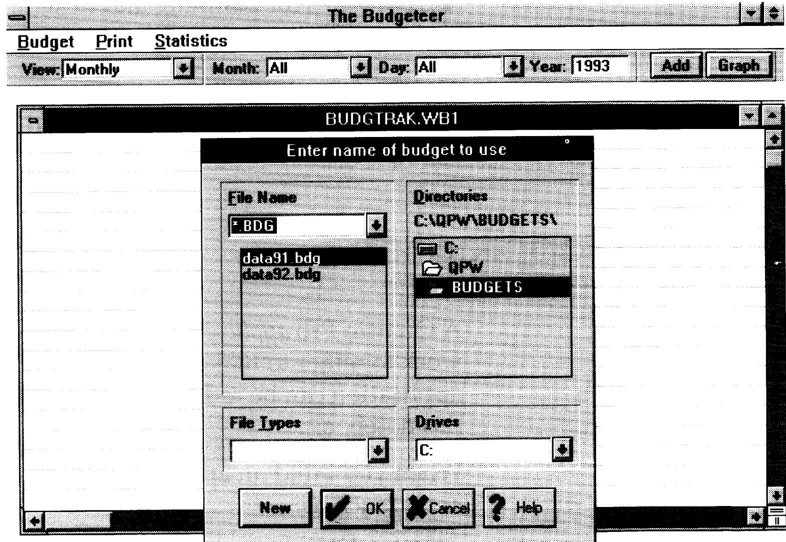


Figure 5.1 The opening screen in the Budgeteer

You can see how different this application's *user interface* (UI) is from the standard Quattro Pro UI. The screen differs from Quattro Pro in these ways:

- `The Budgeteer` appears in the title bar.
- A custom dialog box appears onscreen.
- The names of menus on the menu bar are new.
- A custom SpeedBar resides below the menu bar.
- The notebook window doesn't contain page tabs, row and column borders, or vertical gridlines.

The Budgeteer application is explained in the next section, so that you can explore it as a user.

Using the Budgeteer

The Budgeteer manages a set of databases stored in a separate notebook. This notebook is called a *budget file* and is created by the Budgeteer. Each database in the budget file stores a set of expenses. Each database contains expenses that occur at a certain interval. For example, if you (as a user) pay your car insurance annually, that expense is stored in the Annual database; monthly expenses like groceries are stored in the Monthly database, and so on. The Budgeteer lets you set up an expense-tracking system *without* having to understand databases.

After loading a budget file (using Budget | Open or the opening screen of the Budgeteer) a *record window* is displayed showing the records in the database you're working on (by default, the Monthly expense database is displayed). If the database being viewed is empty (or you've created a new budget file using the opening screen or Budget | New), the window is empty. To specify the database to work on, use View on the SpeedBar.

You can use the Budgeteer SpeedBar to specify the expense database to work with and to search for specific records in that database. Commands on the Budgeteer menu bar let you save, load, and print budget files. For an overview of the Budgeteer menus and SpeedBar, see page 288. The remainder of this section provides instructions to you as the user for performing basic Budgeteer tasks.

Creating a new budget file

To create a new budget file,

1. Load the Budgeteer and choose New from the initial dialog box. This displays an empty record window. (You can also choose Budget | New if the Budgeteer is already running.)
2. Use Budget | Add Expense to add expenses to the databases in this budget file (discussed next).
3. Use View and Add on the Budgeteer SpeedBar to add records to the databases in this budget file.
4. Choose Budget | Save As, then enter a name for the budget file.
5. Choose OK to save the budget file. By default, budget files are saved with the file extension BDG.

Adding expenses to a database

You can use Budget | Add Expense to add expenses to any database in the active budget file. To add an expense,

1. Choose Budget | Add Expense. This displays the dialog box shown in Figure 5.4 (on page 292).
2. Click a button in the Frequency group box to choose the name of the database you want to add an expense to. By default, the database being viewed is used.
3. Choose New Item and enter the name of the expense to add. Current Items shows the existing expenses.
4. Choose OK to add the expense to the database.
5. Save the budget file using Budget | Save or Budget | Save As.

Adding records

To add records to a database,

1. Use View on the Budgeteer SpeedBar to specify the database you want to add a record to.
2. Choose Add from the Budgeteer SpeedBar. This displays a block containing the names of all expenses in the database. Date is also added to specify the entry date of the record; the Budgeteer uses this date when searching databases. It defaults to the current system date.
3. In the cell below each expense name, enter the cost of that expense. You can use arrow keys to move between the expenses.

4. To add the record to the expense database, press *Enter* (with nothing on the input line) or *Esc*.
5. Repeat the previous three steps for each record you want to add.
6. Use Budget | Save As or Budget | Save to save your changes.

Viewing an expense database

The Budgeteer displays a record window that shows the records in an expense database. You can use the Budgeteer SpeedBar to view a different database (using View) or reduce the number of records that display (using Day, Month, and Year, discussed next). You can use the scroll bars to scroll through records.

To view expense databases in a different budget file, use Budget | Open to load the budget file containing the databases.

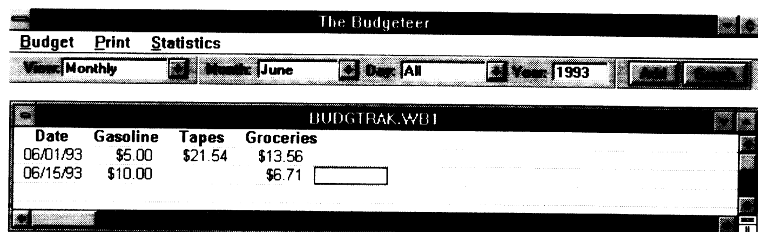
Searching for specific records

By default, the Budgeteer shows you all the records in the database specified by View. You can use Month, Day, and Year to view records entered during a specific time period; if you don't want one of these controls to affect the search, set that control to All. For example, to view only records entered in June of 1993,

1. Set Month to June.
2. Set Day to All to ensure that it doesn't affect the search.
3. Set Year to 1993.

The next figure shows an example of how the previous query might affect the active budget file.

Figure 5.2 Searching for specific records



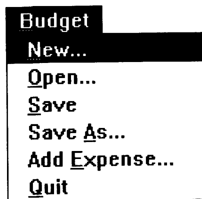
The Budgeteer automatically updates the record window to display only those records that meet the search criteria specified in the SpeedBar. You can graph the information in these records (using Graph on the SpeedBar), print them in a report (using Print | Current), or display statistics about them (using Statistics).

The Budgeteer UI

This section provides an overview of the Budgeteer menus and SpeedBar. For more task-oriented information, see the previous section.

The Budgeteer menus

The Budgeteer menu bar provides commands to perform global operations like saving budget files, printing, and displaying statistics. For example, here's what the Budget menu commands do:



- **New** creates a new budget.
- **Open** loads an existing budget file.
- **Save** writes current data to the budget file.
- **Save As** writes current data to a different budget file.
- **Add Expense** creates a new expense column. You can use this command to add expense categories to any database in the active budget file, not just the database being viewed.
- **Quit** exits the application and restores standard Quattro Pro settings.

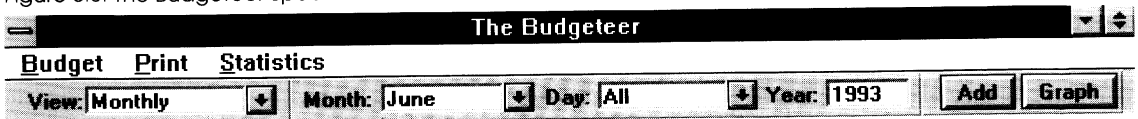
You can use commands on the **Print** menu to print the contents of the active budget file. You can use Print | Current to print the records currently displaying in the record window, or All to print the contents of all expense databases in the active budget file.

The **Statistics** command displays statistics on the records displaying in the record window, including the total expense cost, average expense cost, greatest cost, and smallest cost.

The Budgeteer SpeedBar

When the Budgeteer is active, the standard Quattro Pro SpeedBar is replaced by the Budgeteer SpeedBar, which lets users display the precise view of the data they need.

Figure 5.3: The Budgeteer SpeedBar



Here's what the SpeedBar options do:

Remember, a budget file contains multiple expense databases. Use View to specify which database you're working on.

- **View** specifies the database to display in the record window. You can use Month, Day, and Year to search for and display specific records.
- **Month** lets you display records entered during a specific month. For example, set it to June to display any records entered for June. To view all months, set Month to All.
- **Day** lets you display records entered on a specific day of the week. For example, set it to Tuesday to display any records entered on a Tuesday. To view all days, set Day to All.
- **Year** lets you display records entered during a specific year. For example, set it to 1994 to display any records entered during 1994. To view all years, set Year to All.
- **Add** adds a new record to the expense database being viewed. Remember, a new record appears only if it meets the search conditions specified by the SpeedBar.
- **Graph** creates a graph using the records in the record window. Each expense is plotted as a separate series.

Looking at how the application works

When you finish exploring the Budgeteer as a user, choose Budget | Quit to close the application. Notice that the standard Quattro Pro UI is restored. To see the macro does this, see Figure 5.7 on page 296.

You can stop the Budgeteer's autoload macro before it runs to study the application as a developer and see how it works. To stop the autoload macro,

1. Choose Tools | Macro | Debugger.
2. Open BUDGTRAK.WB1 using File | Open or File | Retrieve.
3. Choose Terminate from the debug window's menu bar to stop the autoload macro that runs the Budgeteer.

The Budgeteer notebook is ready for your review. One advantage of notebooks is that you can organize the components of your application. The following table lists some of the pages of Budgeteer notebook and their purpose.

Table 5.1
Pages used in the Budgeteer

Page(s)	Purpose
Startup	This page contains the macro that changes the Quattro Pro UI and runs the Budgeteer. It also contains the macro that closes down the Budgeteer.
Menus	This page contains the menu block that defines the Budgeteer menu bar.
Help	This page contains help messages displayed by the Budgeteer.
*Task*_Macros	These pages contains macros specific to a Budgeteer function. For example, `ChangeDB_Macros` contains macros to change expense databases; `Query_Macros` contains the macros to search expense databases.
Expense_View	This page is the record window that the user sees.
Stats	This page is the window that the user sees when they choose Statistics.
Input	This page contains the form that users manipulate to add records.
Daily...Annually	These pages contain the expense databases when the Budgeteer is running. Each page contains one database.
Graphs	This page contains the dialog boxes used by the Budgeteer. (Discussed next.)

The Budgeteer uses several dialog boxes, as listed in the next table.

Table 5.2
Budgeteer dialog boxes

You can study the Budgeteer SpeedBar by loading BUDGTRAK.BAR, a file in your Quattro Pro program directory.

Dialog Box	Purpose
AddExpense	Displayed by Budget | Add Expense. It lets the user view expenses in a specific database and add a new expense.
InitOpen	Displayed by many commands on the Budget menu. It's used to get the name of a file to load or save. The title in this box is often changed by macro commands. The New button is hidden by macro commands when it's not needed.
Warn	Displayed by many Budgeteer commands. It's used to confirm some action the user is about to perform. The warning message is often changed by macro commands.

Many of the macro commands in the Budgeteer notebook have comments listed to their right. These comments help explain how the Budgeteer works.

Understanding application components

In a Quattro Pro application, each component is used for a specific purpose:

- **Dialog Boxes** prompt users for information and let the user make choices.
- **SpeedBars** give users shortcuts to common application tasks and perform operations that affect the active data. They also display information about the active window or object.
- **Menus** provide custom lists of operations the user can choose from the menu bar. Menu commands usually perform more global operations (like saving a file).

You use macros to assemble these components into an application. You can also use macros to automate Quattro Pro operations, change Quattro Pro's appearance, and display dialog boxes.

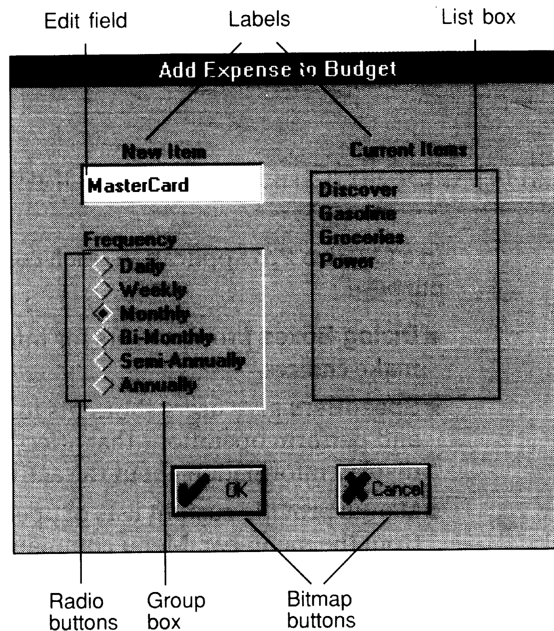
You can also use *link commands* to assign actions or connections to controls in a dialog box or a SpeedBar. A link command can get or change an object's property settings, or run macros.

Designing dialog boxes

When you want to create a dialog box (or a custom SpeedBar) you can choose Tools | UI Builder to work in a dialog window. When a dialog window is active, the SpeedBar changes to provide tools that create dialog *controls*, objects you can create in a dialog box or a SpeedBar.

Users can use controls to set options and initiate actions. The following figure shows some examples of controls in a dialog box:

Figure 5.4
A dialog box in the Budgeteer

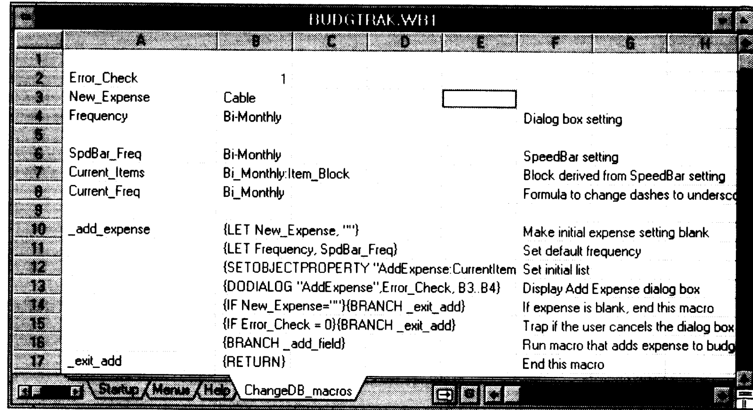


Displaying dialog boxes

After you create a custom dialog box, you can display it using the macro command {DODIALOG}. The most important argument in {DODIALOG} is the third argument, which is a block that contains the initial settings of the dialog box controls. See page 329 for more information on displaying dialog boxes.

The following figure shows the macro in the Budgeteer that displays the Add Expense dialog box (shown in Figure 5.4). In this macro, information the user enters in the dialog box is passed to the Budgeteer application:

Figure 5.5
A macro that uses {DODIALOG} to display a dialog box



Using link commands

Link commands perform dynamic actions in response to *events*, which are things the user does in an application. For example, a link command can be assigned to a control so it will get an object property when the control is clicked, or display additional controls.

Events recognized by Quattro Pro include:

- clicking
- right-clicking
- pressing a key
- entering a new value

Link commands also let you quickly connect a control to another control or a notebook cell.

For a complete list of Quattro Pro's application events, see Table 6.19 on page 354. To learn how to create link commands, see page 352.

Designing SpeedBars

When specific tasks are frequently repeated in an application, you can create a custom SpeedBar to provide user shortcuts. SpeedBars are also suited to operations that affect the current data, and can display information about the current data. For example, the Budgeteer SpeedBar contains common database operations such as defining a data query and graphing the data; it also displays the name of the database being viewed.

A SpeedBar is a special type of dialog box; here's how SpeedBars are different from dialog boxes:

- SpeedBars appear only at the top of the Quattro Pro window.
- Users can't use buttons to close them (unless you add a control to do this).
- Titles aren't displayed in SpeedBars.
- SpeedBars are saved as separate files with a .BAR extension.
- The {DODIALOG} macro command can't display a SpeedBar.

Displaying SpeedBars

After you create a custom SpeedBar, you can use the SpeedBar property (in the application Object Inspector) or the macro command {Application.SpeedBar} to display the SpeedBar in an application. You can either display a custom SpeedBar below the standard Quattro Pro SpeedBar, or by itself, without the standard Quattro Pro SpeedBar.

Displaying a custom SpeedBar by itself is a two-step process. First you hide the standard Quattro Pro SpeedBar, then you display your custom SpeedBar. See Chapter 6 for details on creating, removing, and displaying SpeedBars.

Designing menus

The operations performed by SpeedBars often overlap with the operations performed by menus; try to use menus for operations that affect the data in a global way (like saving a file, printing a database, and so on).

You can add custom menus and menu commands to the menu bar and replace the menu bar with one command.

Menu commands can

- initiate actions
- run macros
- display dialog boxes
- display other menus

To create a menu, you describe the menu with a block of labels. The following table lists macro commands you can use to modify menus, menu commands, and the menu bar:

Table 5.3
Macro commands for defining a custom menu

Macro command	Description
{DELETEMENU}	Removes a menu from the current menu bar (including submenus like Tools | Macro).
{DELETEMENUITEM}	Removes menu commands from the specified menu.
{ADDMENU}	Adds names to the current menu bar.
{ADDMENUITEM}	Adds menu commands to the specified menu.
{SETMENUBAR}	Replaces the active menu bar with the menu described by the block.
{SETOBJECTPROPERTY}	Changes existing menu commands.

For example, the following figure shows the Budgeteer's menu bar, and some of the labels that define it. The Budgeteer uses {SETMENUBAR} to replace the standard menu bar with the items described in this block.

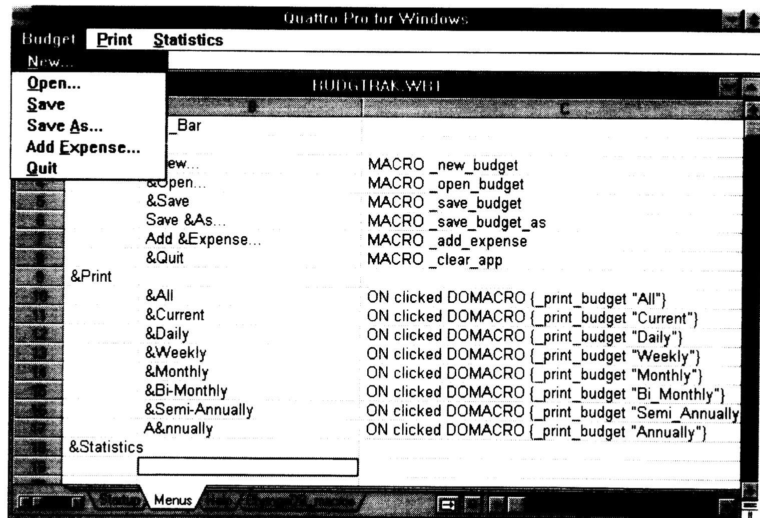


Figure 5.6
The Budgeteer menu definition

Notice Column C in the previous figure; each menu command has one or more link commands in its block. These link commands determine what action occurs when the menu command is chosen. Command names that appear in column B appear as items on the menus defined in column A. See Chapter 7 for more information on these link commands.

Designing macros

Macros are the heart of every application; they're the primary method you use to pull all your application components together.

In addition to the {DODIALOG} macro command, which displays dialog boxes, macros are most commonly used in an application to

- automate tasks
- duplicate keyboard and mouse actions
- change the active menus
- affect the screen display
- select, reposition, and resize objects
- prompt users for input

The remainder of this section lists some of the uses of macros in an application.

Autoload macros

Developers typically use an autoload macro (see page 115) to set up the application and present a unique look for the application. For example, the autoload macro in the Budgeteer hides parts of the standard Quattro Pro UI, changes the menu bar to the Budgeteer menus, and displays the initial Budgeteer dialog box.

Figure 5.7
The Budgeteer's autoload and cleanup macros

BUDGTRAK.WB1

	A	B	F
1	\0	{Application.Title "The Budgeteer"}	Set application title
2		{Application.SpeedBar "BUDGTRAK.BAR"}	Set application SpeedBar
3		{Application.Display "None,No,No,No,A:A1..B:B2"}	Hide Input Line, Status Bar, and SpeedBar
4		{Application.Enable_Inspection "No"}	Disable right-clicking
5		{Page.Borders "No,No"}	Hide row/col borders
6		{Notebook.Display "Yes,Yes,No"}	Hide page tabs
7		{SETMENUBAR menu_block}	Set menu bar to application menu
8		{BRANCH _init_expense_view}	Display expenses
9			
10	_clear_app	{Application.Title "Quattro Pro for Windows"}	
11		{Application.SpeedBar ""}	
12		{Application.Display "None,Yes,Yes,Yes,A:A1..B:B2"}	
13		{Application.Enable_Inspection "Yes"}	
14		{Page.Borders "Yes,Yes"}	
15		{Notebook.Display "Yes,Yes,Yes"}	
16		{SETMENUBAR}	
17		{FileClose 0}	

Startup / Menus / Help / ChangeDB_macros

Changing properties with macro commands

Quattro Pro offers numerous macro commands that developers can use to modify the standard look of Quattro Pro or to change property settings.

Some of these macro commands are

- {Application.Display}
- {GETOBJECTPROPERTY}
- {GETPROPERTY}
- {Notebook.Display}
- {Page.Borders}
- {Page.Grid_Lines}
- {SETOBJECTPROPERTY}
- {SETPROPERTY}

See Chapter 4 for more information on these commands.

Preparing to build an application

As a developer, it can be hard to resist jumping right in and building an application. But before doing this, take the time to carefully plan and design the application. This approach makes the application easier to debug and revise.

Planning an application

Find out what users want and need in an application.

As a developer, consider these fundamental issues:

- **User needs.** Start by determining what tasks users need to accomplish. Encourage users to suggest ways to make the proposed application useful to them.
- **Examples.** Find existing features or applications in other products that perform tasks similar to the ones you plan to design. Ask users if they can think of any way to improve these examples.
- **Simplicity.** Try to make the application as easy to use as possible. Try to provide help messages and help lines for users within the application.

Design guidelines

Encourage users to share their comments throughout the development process.

As a developer, consider using the following tips:

- **Involve users.** Let users review your work on the application when early working versions are ready. Use their suggestions to improve and refine the design.
- **Think small.** Keep all macros, dialog boxes, and menus for a single application organized in one notebook. Also, try to limit the number of supporting files (notebooks, SpeedBars, graphics, text, and so on). This way, you'll have fewer files to install and maintain on user's computers.

- **Be consistent.** Define and follow a set of rules for naming and storing components of your applications. For example, you could begin all component names with the same three-letter code. Also, keep your application files in a specific directory and instruct your users to do the same.
- **Document your work.** Since you might need to update or revise the application later, be sure to include development notes in the application. For example, you can add specific comments and labels to macros, or general comments for an entire notebook page of menu items.
- **Protect yourself.** Limit user access to critical components of the application by hiding UI elements such as the tab cluster or by protecting cells from editing (see Chapter 4 of the *User's Guide* for details). This will keep unexpected maintenance of the application to a minimum.
- **Avoid traps.** If you change the standard menu bar, you might be unable to run a macro by choosing Tools | Macro | Execute. When developing applications, run Quattro Pro in Developer mode (discussed next); then you can press *Ctrl+Shift+N* to restore the standard menu bar and reveal hidden UI components. You can also write a macro to restore Quattro Pro to its original state and assign the macro to a key (see page 113). Handy macros like this one can be saved to a macro library, and the macro library can be open when testing or developing.
- **Clean up.** Remember to restore the standard Quattro Pro settings after users exit your application. For an example of a macro that does this, see Figure 5.7 on page 296.

Using the Developer mode

Starting Quattro Pro with the command line switch `/D` runs Quattro Pro in Developer mode. Developer mode adds special properties to Object Inspector menus and lets you use a shortcut key, *Ctrl+Shift+N*, as described in the following tables.

Table 5.4
Accessing properties in Developer mode

You don't have to be in Developer mode to change or read these property settings; macro commands can always access them.

Menu name	Property	Description
Application	Enable Inspection	Lets you disable right-clicking, the Property menu, and the Object Properties key (*F12*). Use this to disable Object Inspector menus.
Application	Title	Lets you change the text that displays in the Quattro Pro title bar.

In Developer mode, all drawn objects in a graph window gain the following properties.

Table 5.5
Special graph object properties

Property	Description
Dimension	Lets you move and resize graph objects created with the SpeedBar in the graph window.
Name	Used by macro commands, link commands, and @functions to read or change property settings.

For a list of all Quattro Pro object properties (including more hidden properties), see Appendix B.

Table 5.6
Pressing *Ctrl+Shift+N*, the Developer mode shortcut key

State of program	Description
Debugging a macro	Stops macro execution and exits Debug mode.
All other states	Restores the standard Quattro Pro menu bar, status line, input line, and SpeedBar. Also sets the Enable Inspection property back to Yes, which re-enables Object Inspector menus.

Where do I go from here?

All topics introduced in this chapter's conceptual overview are covered in greater detail in this manual. Here's where to find the information you need:

- **Macros (general).** For details on general rules for using macros, such as syntax, arguments, and so on, see Chapter 3.
- **Macro commands.** Information about specific macro commands, such as {DODIALOG}, is listed alphabetically in Chapter 4.

- **Components and link commands.** Chapter 6 contains a full discussion of how to create dialog boxes, SpeedBars, and link commands. Chapter 7 discusses link commands in menus.
- **Menus.** For more information on changing menu command properties, creating menus, or creating menu commands, see Chapter 7.

C H A P T E R

6

Dialog boxes and SpeedBars

Quattro Pro has several features you can use to create custom notebook applications, including macros, SpeedBars, and dialog boxes.

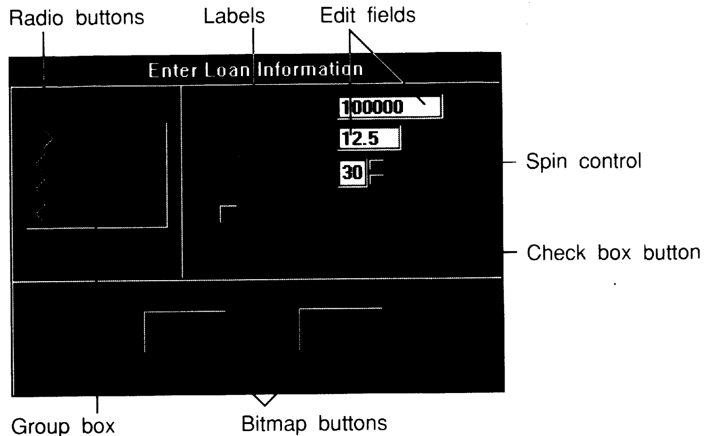
This chapter addresses "you" as the developer of an application. The "user" is someone who uses a completed application.

This chapter describes how to create custom dialog boxes and SpeedBars. Through the process of building a sample dialog box (similar to LOANPMT.WB1), you'll learn the basic steps for building a new dialog box, adding controls to it, and assigning link commands to each control.

An overview

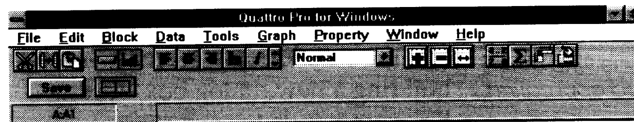
A *dialog box* is a box of options that you create to prompt the application user for information. It can contain various *controls,* such as check boxes and edit fields (as shown in the following figure).

Figure 6.1
A sample dialog box



A *SpeedBar* is a custom tool you create that remains onscreen while an application runs (just as the Quattro Pro SpeedBar does). It gives the user shortcuts for commonly used commands and procedures.

Figure 6.2
A sample SpeedBar (below the standard SpeedBar)



Dialog boxes are saved with the notebook; SpeedBars are saved as separate files.

From a user's point of view, the main difference between dialog boxes and SpeedBars is that dialog boxes appear at specific times during an application; SpeedBars usually remain onscreen while the application runs.

Because of this difference, you'll tend to put commonly used commands or actions on a custom SpeedBar, and reserve dialog boxes for those places in an application where you need specific information from a user.

Tools for building dialog boxes

Dialog boxes and SpeedBars are similar in many ways; you use similar commands and tools to build them. Before you start to build your first dialog box, you should be somewhat familiar with the commands and tools you'll be using.

The dialog window

When you choose Tools | UI Builder, you see an empty dialog box within a dialog window.

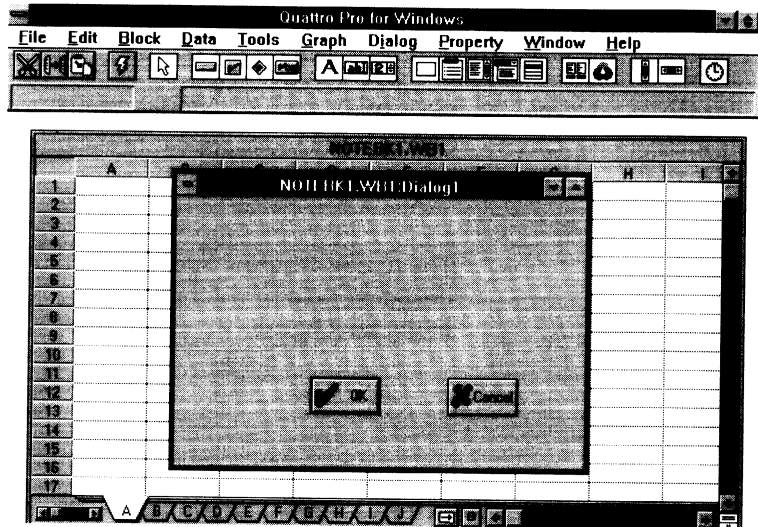


Figure 6.3
A new dialog window

New dialog boxes contain an OK button and a Cancel button, to provide a consistent way for the user to close the dialog box. Also, a default name is automatically assigned.

When you're creating a dialog box in a dialog window, it looks slightly different from how it will appear to the user in a completed application.

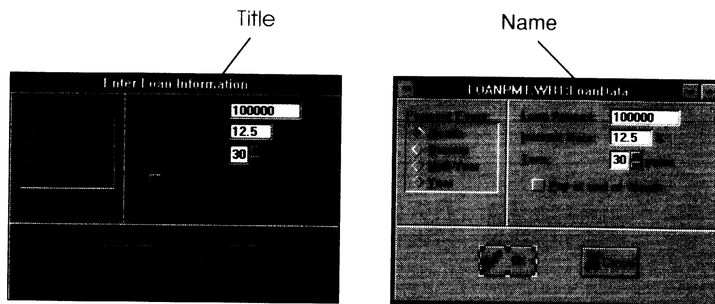


Figure 6.4
A dialog box as the user sees it (left) and as the developer creates it (right)

The *dialog window* is the dialog box in an editable form; it contains all the Windows user interface elements for a window: a window border, a menu-control box, and Minimize and Maximize buttons.

In the dialog window, the *name* of the dialog box (LOANPMT.WB1:LoanData) appears in the title bar. You'll use this name in a macro to display the dialog box or change its property settings.

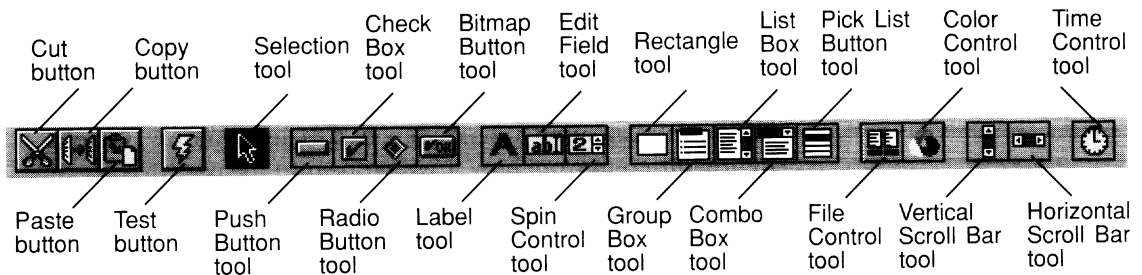
This name is different from the *title* of the dialog box (Enter Loan Information), which is what the user sees when the application runs.

SpeedBars don't have titles, but they do have names (which, like dialog box names, appear in the title bar).

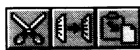
The dialog window SpeedBar

The dialog window SpeedBar contains tools that are specifically designed to help you build dialog boxes and SpeedBars, as shown in the next figure.

Figure 6.5: The dialog window SpeedBar



These tools let you quickly create, copy, move and test *controls*. Controls are elements you put in a dialog box to gather information from the user or to perform a desired action.



The **Cut**, **Copy**, and **Paste** buttons are shortcuts for Edit | Cut, Edit | Copy, and Edit | Paste, respectively. You can copy a control to the Clipboard to duplicate it or to move it to another dialog box or SpeedBar.

The **Test button** tests the operation of a dialog box (or SpeedBar). In Test mode, the dialog box controls behave exactly as they will when the application runs.

The **Selection tool** lets you select a control in the dialog window to move it, resize it, change its properties, and so forth. You can select several controls at a time to move or size them together.

The remaining tools on the SpeedBar let you add controls to a dialog box; they are described in detail later in this chapter.

General procedures

This section shows the basic steps you'll follow to create a custom dialog box or SpeedBar. You'll follow these steps in the next section, to build a sample dialog box and a sample SpeedBar.

Procedures for creating a dialog box

Here are the basic steps to follow to create a dialog box:

1. Choose Tools | UI Builder. A dialog window appears.
2. Enter a name for the dialog box. (Later, you'll use this name in a macro to make the dialog box appear.)
3. Enter a title for the dialog box. (The title is what the user will see.)
4. Add controls to the dialog box.
5. Customize each control, if necessary.
6. Resize the dialog box, if you want.
7. In the notebook, create a {DODIALOG} command to call the dialog box and pass initial defaults to it.
8. Add a link command to each control, if necessary. (A link command indicates what should happen when a user manipulates that control in a specific manner.)
9. Test the operation of each control.
10. Save the dialog box (by saving the notebook it is in).

Procedures for building a SpeedBar

Here are the basic steps to follow to create a SpeedBar:

1. In the notebook, select the Graphs page (click the SpeedTab button).

2. Choose the New SpeedBar tool from the Graphs page SpeedBar. A long, narrow dialog window appears—this is the blank SpeedBar.
3. Add controls to the SpeedBar.
4. Customize each control as necessary.
5. Add a link command to each control to indicate what should happen when the user clicks this control.
6. Test each control to make sure that it works properly.
7. Save the SpeedBar. (Since SpeedBars are saved as separate files, you can use a SpeedBar with different notebooks.)
8. Display the SpeedBar, using the SpeedBar property in the application Object Inspector.

Creating a sample dialog box

You'll follow a slightly different process to create a new SpeedBar; see page 318.

Suppose you want to create a dialog box similar to the one shown in Figure 6.1 from the sample file LOANPMT.WB1. It will gather loan information from the user, then calculate a monthly payment from that information. (Before you proceed with the steps in this chapter, you may want to load the LOANPMT.WB1 file and follow the instructions, to see what you'll be creating.)

You want the user to indicate three things: the car loan amount, the term, and whether the payment falls on the first of the month or the 15th of the month. (For the purposes of this example, we'll assume an interest rate of 12.5%.) From this information, a payment amount will be calculated and placed in a cell in the notebook.

Opening a new dialog window

Let's start building the dialog box by opening a blank dialog box and giving it a name and a title.

There are two ways to reach the dialog window. Typically, you'll choose Tools | UI Builder from within a spreadsheet page. (You could click the dialog icon from the Graphs page instead.)

1. Starting from a new notebook, choose Tools | UI Builder. You'll see a new dialog window.

2. To give this new dialog box a name, right-click the dialog box background, then choose Name.

   Type `LOANDB` and choose OK. Notice that the title bar of the dialog box shows its new name.

3. Next, create the dialog box title. Right-click the dialog box background, then choose Title.

   Type `Enter Loan Data` and choose OK. (You won't see any visible change here; the title is what a user sees.)

4. Dialog boxes are stored within their notebook. To save this dialog box, save its notebook—click anywhere in the notebook, choose File | Save As, then enter `CARLOAN`.

There is an alternative way to create a dialog box: From the notebook, choose the Graphs page, then select the New Dialog tool from the Graphs page SpeedBar.

Adding controls to the dialog box

Now that you've named the dialog box and given it a title, the next step is to add the desired controls to it.

Quattro Pro offers a variety of different controls; each control is best suited for presenting (or receiving) a certain type of information. Each control displays a setting and provides a way for users to change that setting.

Following is a brief description of the most commonly used controls (full descriptions of controls begin on page 340).

Push Button. A push button usually performs a specific action when the user clicks it. You, the developer, determine what that action is.

Check Box. Check boxes present a Yes/No choice to the user. The user checks the check box to accept the choice. Check boxes are often placed within group boxes to present the user with mutually exclusive choices.

Radio Button. Radio buttons (also called *option buttons*) are usually grouped, to give the user a mutually-exclusive list. When a user clicks a radio button, its diamond darkens to indicate that it's chosen.

Label. A label clarifies for the user what a specific control does.

Edit Field. An edit field is where a user types specific information.

Spin Control. A spin control lets the user click an arrow to increase or decrease the value it displays. (The user can also type a value instead.)

Rectangle. A rectangle embellishes a dialog box or groups two or more controls together (see page 335).

Group Box. A group box usually contains other controls such as radio buttons or check boxes. It looks like a rectangle with a title at the top.

To add controls to the sample dialog box LOANDB, follow these steps:

1. First, go back into the dialog window: Select the Graphs page and double-click the LOANDB icon.

   This illustration shows the dialog box you'll be creating. Refer to it when positioning new controls.

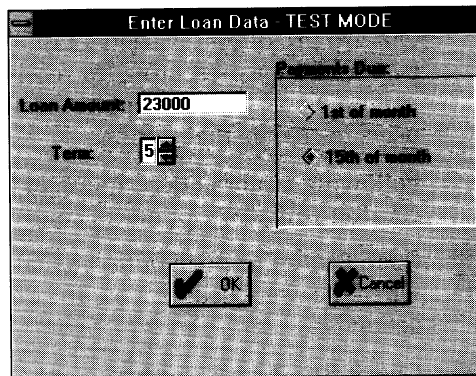


Figure 6.6
The finished LOANDB dialog box

2. Create an edit field, where the user can enter the amount of the desired loan: Choose the Edit Field tool from the dialog window SpeedBar.
3. Position the pointer near the upper left corner of the dialog box (leave room for a label on the left). Click to drop the edit field into place.

   The control appears, with its top left corner at the position you clicked.

4. To make the control a little bigger, drag one of the edit field's lower corners to the right (or left).

You can create a control and size it all in one step: Instead of clicking the mouse button, move to the position where you want the control to start, then drag the pointer until the control is the desired size.

You'll customize this edit field later. Next, however, add a spin control to the dialog box, where the user can specify the term of the loan.

1. Choose the Spin Control tool from the dialog window SpeedBar.

2. Position the pointer beneath the edit field, then click to drop the spin control into place.

Your dialog box should look similar to this:

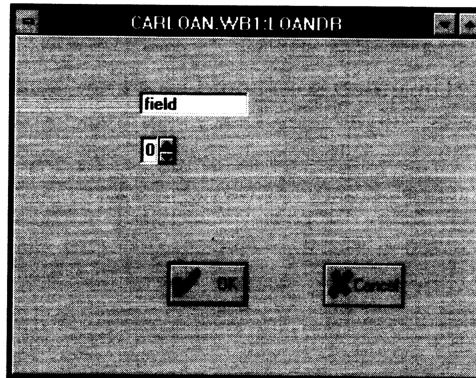


Figure 6.7
The LOANDB dialog box after adding an edit field and a spin control

You'll specify the range of values for the spin control later. Next, add some radio buttons where the user can specify whether the payment should fall on the first day of the month or the last.

1. Choose the Group Box tool from the dialog window SpeedBar.
2. Place the pointer near the upper right corner of the dialog box, then drag toward the middle of the dialog box. (Refer to Figure 6.6 on page 308 for size and position guidelines.)
3. Release the mouse button; the group box drops into place.

   The LOANDB dialog box should look similar to this:

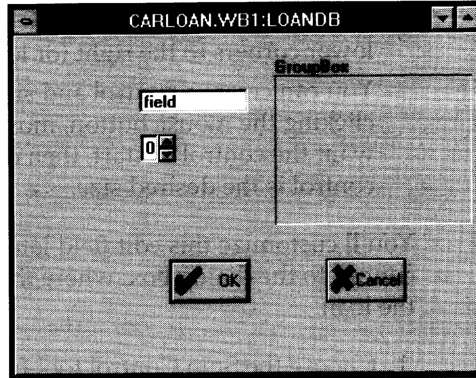


Figure 6.8
The LOANDB dialog box after adding the group box

4. To add the radio buttons to the group box, choose the Radio Button tool from the dialog window SpeedBar.
5. Click inside the group box near the upper left corner to drop the first radio button into place.
6. Repeat steps 4 and 5, placing the second radio button just below the first one.

The group box should now look like this:

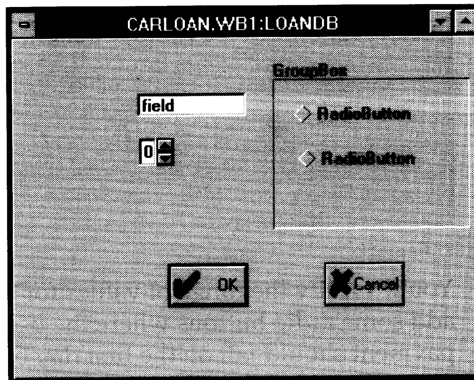


Figure 6.9
The group box after adding two radio buttons

By the way, there's a fast way to fill a group box with radio buttons—just hold down the *Ctrl* key while you resize the group box.

Labeling controls

To help explain to the user what these controls do, let's add labels to each control in the LOANDB dialog box:

1. Choose the Label tool from the dialog window SpeedBar.
2. Move the pointer just to the left of the edit field you created, then click. (Move the edit field to the right if you need more room.)
3. Next, add text to this new label: Double-click this label, type `Loan Amount:` and press *Enter.*
4. To add a label to the spin control, choose the Label tool from the dialog window SpeedBar again, move the pointer just to the left of the spin control, then click.
5. Double-click this label, type `Term:` and press *Enter.*

Your dialog box should look similar to the next figure.

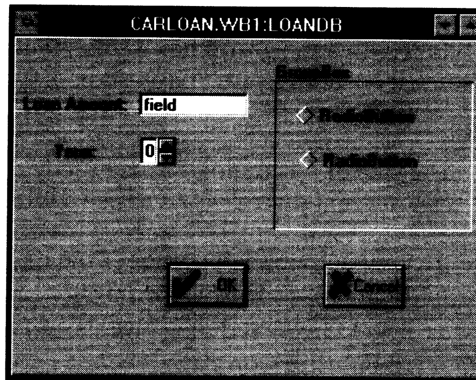


Figure 6.10
The LOANDB dialog box after adding labels

Customizing controls

Next, specify some limits for the edit field and the spin control.

Give the edit field a range of 1000–30000:

1. Right-click the edit field in the LOANDB dialog box. You'll see its Object Inspector.
2. Choose Field Type, then choose Integer.
3. Right-click the edit field again, choose Minimum, type `1000` and choose OK.

4. Right-click the edit field again, choose Maximum, type `30000` and choose OK.

This field will not accept text strings or alphabetical characters, or values above 30000 or below 1000.

Next, limit the spin control to the range 1–5 (one- to five-year loans):

1. Right-click the spin control in the dialog box. Choose Minimum, type `1` and choose OK.
2. Right-click the spin control button again, choose Maximum, type `5` and choose OK.

Next, add descriptive text to each radio button:

1. Double-click the first radio button, type `1st of month` and press *Enter*.
2. Repeat this process for the other radio button: Double-click it, type `15th of month` and press *Enter*.
3. To give the group box a title, double-click the group box title (it currently says "GroupBox"), type `Payments Due:` and press *Enter*.

The size of this dialog box is exactly the size the user will see. You can resize it just as you'd resize a window:

For more precise sizing, use the Dimension property.

1. Grab a lower left corner of the dialog box and drag it upward, then release the mouse button.

   The LOANDB dialog box should now look like the illustration in Figure 6.6 on page 308.
2. Save your work: Click anywhere in the Graphs page, then choose File | Save. Remember, your dialog box is saved with the notebook.
3. To set up an area of the notebook to receive information from the dialog box controls, return to any blank notebook page before you continue.

Using the {DODIALOG} command

The next step is to create a macro that displays the dialog box and stores the user's choices in cells in the notebook. The macro will send initial values to each control in the dialog box. If the user

changes a setting in the dialog box, that change will be sent back to a cell in the notebook when the user closes the dialog box.

First, set up an area that will send default settings to the dialog box:

1. In cells A2, A3, and A4, respectively, enter `Amount:`, `Term:`, and `Due Date:`.
2. Next, enter the initial defaults: In cells B2, B3, and B4, enter `20000`, `4`, and `"15th of month`, respectively.

   This sets the default loan amount at 20,000, the default term at 4 years, and the default payment date to the 15th of the month.

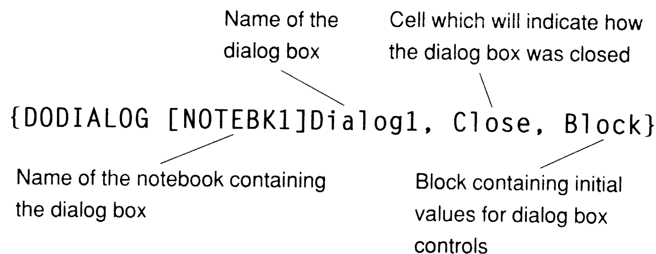
Now set a cell to display the amount of the monthly payment. The formula in this cell will use values the user enters in the dialog box.

1. In cell A6 enter `Payment:`.
2. In cell B6 enter `@PMT(B2,.125/12,B3*12)`.

The result in cell B6 should be 531.60 (set the cell's format to Currency and recalculate, if necessary). This payment amount is based on the loan amount in cell B2 and the term in cell B3.

Next, enter a macro command that will display the dialog box and send the user's choices back to the notebook. You'll use the {DODIALOG} command to do this. The {DODIALOG} command will contain several arguments, which are noted as follows:

Figure 6.11
The arguments in the {DODIALOG} command



- The first argument is the name of the notebook that contains the dialog box you want to display, and the name of the dialog box. The name of the notebook is optional if the dialog box is stored in the active notebook.

- The second argument specifies a cell that will store a value indicating how the user closed the dialog box. If the dialog box was canceled, 0 is stored in the cell; if the user closed the dialog box by pressing *Enter* or OK, 1 is stored.
- The third argument specifies a block that
  - contains initial values for the dialog box controls
  - receives final values from the dialog box controls when the user closes the dialog box

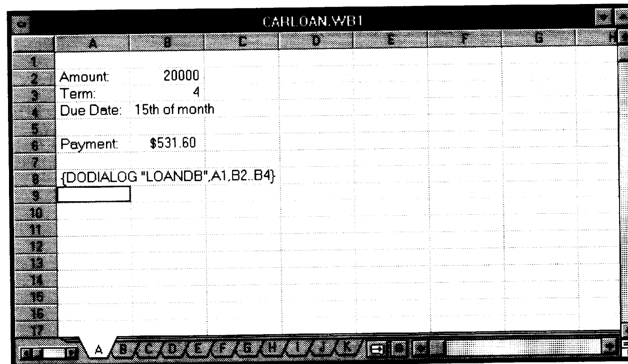
  Each cell in the block (starting at the upper left cell and proceeding row-wise to the lower right cell) sets the initial value of one control. If the user cancels the dialog box, the values in this block remain unchanged.

Enter a macro command to display the LOANDB dialog box:

1. Go to cell A8.
2. Enter `{DODIALOG "LOANDB",A1,B2..B4}`.

The notebook page should now look like this:

Figure 6.12
The notebook after adding the {DODIALOG} command



This command displays the LOANDB dialog box and passes the initial values in cells B2 through B4 to the dialog box controls (20000 for the loan amount, 4 for the term, and 15th for the monthly due date). It also sends a value of 0 to cell A1 if the user cancels the dialog box; otherwise, it sends a value of 1.

To see how the {DODIALOG} command works, try it:

1. Choose Tools | Macros | Execute.
2. Type `A8` and choose OK.

The dialog box appears in the middle of your screen. Try out some of the controls:

1. Click the Loan Amount edit field, then type `15000`.
2. Change the spin control value from 4 to 3 by clicking the down arrow part of it.
3. Choose OK.

Move the dialog box over if you need to.

The settings you chose appear in cells B2 and B3, and the monthly payment calculated from these values is 501.80.

To make this dialog box dynamic, so that the user's changes are immediately reflected in the notebook, let's add link commands to each control.

Linking commands to controls

Link commands are statements you attach to each control to indicate what should happen when the user changes the control.

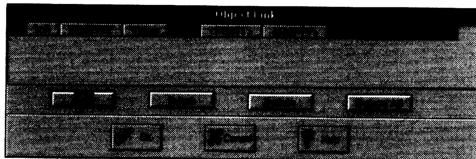
The {DODIALOG} command brings changed values back to the notebook when the user closes the dialog box. But wouldn't it be nice to have a dynamic link so that you don't have to close the dialog box in order to see the calculated loan payment? Let's enter link commands to do just that.

First, add a link command to the edit field to make it write changes back to cell B3 immediately:

If the dialog box no longer appears, switch to the Graphs page and double-click the LOANDB icon.

1. In the LOANDB dialog box, click the Loan Amount edit field.
2. Choose Dialog | Links. The Object Link dialog box appears, as shown in the next figure.

Figure 6.13
The Object Link dialog box



3. Choose Add.

Basically, you enter link commands as statements, specifying "when *a* happens, do *b* to *c*." In our example, the statement will read something like a "what, when, where" statement:

"When the user *changes* a setting, *send* the new *value* back to a specific *cell* in the notebook."

To create this link command, follow these steps:

1. First, specify what *event* triggers the command (a "when" statement): Move to the first button, which is labeled Init. This is the Event pick list button.

   Hold down the left mouse button, drag to Valuechanged, then release. (You want a value sent back to the notebook *when it has been changed*.)
2. Next, specify the *action* that should occur (a "what" statement): Move to the second button, labeled RECEIVE. Hold down the mouse button, drag to SEND, then release. (You want to *send* the change back to the notebook).

   The third button reads Value; since this is correct, leave it as is (you want to send the new *value* back to the notebook).
3. Move to the fourth button, beside TO. This is the Object pick list button, where you specify the object to be acted upon. Hold down the mouse button and choose <POINT>.
4. In the notebook, click cell B2 (*where* you want the value sent). You return to the Link dialog box.
5. Finally, hold down the last button, drag to Value, then release. This indicates *what* you want done to the cell—set its value.

   The Link dialog box should now look like this:

If you're still on the Graphs page, click the SpeedTab button to return to the notebook.

Move the Object Link dialog box out of the way if you need to.

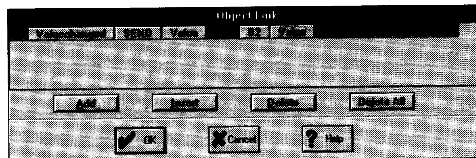


Figure 6.14
The Object Link dialog box after specifying the link command

6. Choose OK.

You'll follow a similar process to add a link command to the spin control:

1. Choose the spin control in the LOANDB dialog box.
2. Choose Dialog | Links, then choose Add.
3. Set the first button (the event) to Valuechanged.
4. Set the second button (the action) to SEND.

   Leave the third button set to Value.

5. Set the fourth button (the object) to POINT, then point to cell B3.
6. Set the fifth button to Value.
7. Click OK to return to the LOANDB dialog box.

Repeat this process for the group box:

1. Choose the group box in the LOANDB dialog box (click the title, to make sure you're not selecting one of the radio buttons by mistake).
2. Choose Dialog | Links, then choose Add.
3. Set the first button (the event) to Valuechanged.
4. Set the second button (the action), to SEND.

   Leave the third button set to Value.
5. Set the fourth button (the object) to POINT, then point to cell B4.
6. Set the fifth button to Value.
7. Click OK to return to the LOANDB dialog box.

By the way, there's a fast way to create a dynamic link; use Dialog | Connect.

Next, test each control to see if the dynamic links work.

Testing the dialog box

To test the link commands:

Choose the Test button from the dialog window SpeedBar (or choose Test from the dialog window's Control menu). The dialog box looks just as the user will see it (except TEST MODE appears next to the dialog box title).

In Test mode, you can't activate any other windows unless you've configured a control in the dialog box to do so.

1. Move the dialog box to the right, so that you can see the cells in the notebook to its left.
2. Enter `23000` for the loan amount. Cell B2 in the notebook should change simultaneously, along with the payment amount in cell B6.
3. Set Term to 5. Cells B3 and B6 should change instantly.
4. Choose OK to end Test mode.
5. To save your work, click any cell in the notebook and choose File | Save.

That's it! In the next section, you'll learn how to make a custom SpeedBar.

Creating a sample SpeedBar

In this section, you'll create a SpeedBar with two buttons. The first button will save the active notebook; the second will draw a box around the active cell or block.

SpeedBars aren't associated with a particular notebook (no icon is created on the Graphs page), so you can build this sample SpeedBar from any notebook.

Opening a new SpeedBar

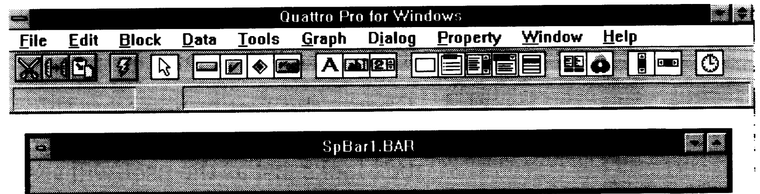
First, open a blank dialog window:

1. Select the Graphs page.

2. Click the New SpeedBar tool on the Graphs page SpeedBar. A dialog window appears; this is the SpeedBar in an editable form.

   Unlike dialog boxes, SpeedBars don't contain an OK and Cancel button.

Figure 6.15
A dialog window for creating SpeedBars



3. Name and save this new SpeedBar (SpeedBars are saved as external files with the extension BAR): Choose Dialog | Save SpeedBar As, type `QUIKSAVE.BAR`, then choose OK. This name appears in the dialog window's title bar.

If you're making a SpeedBar for a specific notebook, consider giving the SpeedBar the same name as the notebook, adding the file extension BAR. This helps you keep track of which SpeedBar belongs with which application.

Adding controls to the SpeedBar

To create the first button,

1. Choose the Push Button tool on the dialog window SpeedBar. Then click near the left end of the QUIKSAVE SpeedBar.
2. Double-click this push button and enter `Save` as the button title.

   The QUIKSAVE SpeedBar should now look like the next figure.

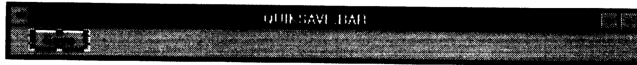


Figure 6.16
The QUIKSAVE SpeedBar after adding a push button

3. To create a bitmap button beside the Save button, choose the Bitmap Button tool on the dialog window SpeedBar, then click to the right of the Save button in the QUIKSAVE SpeedBar.

4. To replace the OK image on this button's face with a small box and grid, right-click it, choose Bitmap, choose `ldrawout` from the list box and choose OK. The button is a little too large to fit in the SpeedBar, so drag one of its top handles down to shrink it, then move the button up.
5. Click the SpeedBar's background to deselect the bitmap button. Then hold down the *Shift* key, click the Save button, and click the bitmap button. Both buttons are now selected.
6. To align these two buttons, choose Dialog | Align.

   This menu offers several commands to help you visually align controls in dialog boxes and SpeedBars.
7. Choose Horizontal Space, type `10` and choose OK. The buttons should now be evenly spaced. Choose Dialog | Align | Top to line up the tops of the controls.
8. Click the SpeedBar's background to deselect the objects. The QUIKSAVE SpeedBar should now look similar to the next figure.

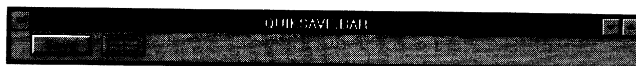


Figure 6.17
The QUIKSAVE SpeedBar after adding a bitmap button

Linking commands to the controls

You can add links to these controls to make the first button save the active file and make the second button draw a box.

1. Select the Save button and choose Dialog | Links. Then choose Add to create a link command.
2. First, indicate *which event* should initiate an action—in this case, a mouse click by the user.

   Pull down the event pick list (which currently reads Init) and choose Clicked.
3. Next, specify *what will happen* when the user clicks the button: a macro should run.

   Pull down the action pick list (which currently reads RECEIVE) and choose DOMACRO.
4. Then enter the macro commands that should run when the button is clicked. The remaining pick list buttons change to an edit field, where you enter the macro command you want this button to perform.

   Type `{FileSave Backup}` in the edit field. This macro will save the file and create a backup of the previous version.

   The Links dialog box should now look like the next figure.

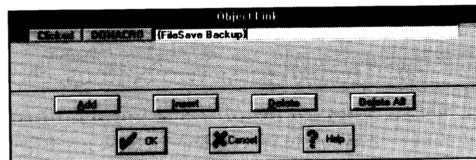


Figure 6.18
A link command that will save the file

5. Choose OK to save the link command and return to the dialog window.

Now enter the link command for the bitmap button, using a similar process:

1. Select the bitmap button, choose Dialog | Links, then choose Add.
2. To specify that a mouse click should initiate the link command, set Event to Clicked.
3. To indicate the action that should happen when the user clicks the button, pull down the action pick list (RECEIVE appears

on it) and choose SET (we'll *set* the linedraw property of a block).

See Appendix B for the syntax for other block properties you can set.

4. In the edit field type:

```
Thin,Thin,Thin,Thin,NoChange,NoChange
```

5. Specify that the link command will change a property in the active block when the button is clicked: Pull down the object pick list (<POINT> appears on it) and choose Active_Block.
6. To indicate that this setting should go to the active block's Line Drawing property, pull down the last pick list and choose Line_Drawing.

   The dialog box should now look like the next figure:

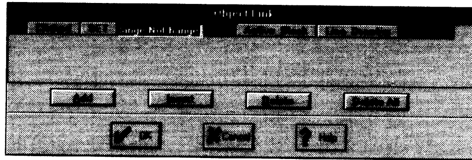


Figure 6.19
A link command that will draw a box around the active block

7. Choose OK to save the link and return to the SpeedBar.

Displaying the SpeedBar

Now that the QUIKSAVE SpeedBar is finished, save it, then use the SpeedBar property in the application Object Inspector to add it to the Quattro Pro window.

1. Choose Dialog | Save SpeedBar.
2. Close the SpeedBar and activate a notebook window.
3. Right-click the application title bar, then choose SpeedBar.
4. Type `QUIKSAVE.BAR` in the edit field, then choose OK. The QUIKSAVE SpeedBar now appears below the standard SpeedBar, as shown in the next figure.

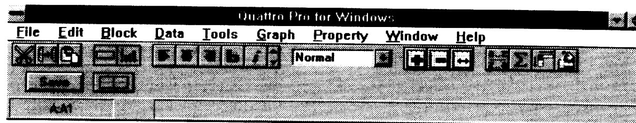


Figure 6.20
The QUIKSAVE SpeedBar as it appears in the notebook

To see how this SpeedBar works, select a block in the active notebook, then click the bitmap button. A box surrounds the block.

To remove the SpeedBar, choose the SpeedBar property again and choose Reset.

Now that you've finished the sample SpeedBar, you can use it with your own notebooks, or add additional controls to it if you like.

The next section describes how to edit and display custom SpeedBars.

More about SpeedBars

To edit a SpeedBar, choose Tools | UI Builder, then choose Dialog | Open SpeedBar. After making any changes, choose Dialog | Save SpeedBar.

Or go to the Graphs page, click the New SpeedBar tool, then choose Dialog | Open SpeedBar, enter the name of the SpeedBar, and choose OK.

When a SpeedBar is in Test mode, you can activate other windows to test the SpeedBar's operation. Also, in Test mode SpeedBars aren't locked to the top of the Quattro Pro window. You can use Test mode to create "floating" SpeedBars. Unlike dialog boxes, multiple SpeedBars can be in Test mode at the same time.

The application Object Inspector (right-click the Quattro Pro title bar or choose Property | Application) has two properties that control the display of SpeedBars:

- Use the **SpeedBar** property to display a SpeedBar below the standard Quattro Pro SpeedBar. Just enter the name of the SpeedBar file. To remove the SpeedBar, choose the Reset option.
- Use the **Show SpeedBar** option of the Display property to specify whether the standard Quattro Pro SpeedBar appears. To hide it, uncheck the Show SpeedBar check box.

Or you can display a SpeedBar with the following macro command:

{Application.SpeedBar *SpeedBarName*}

SpeedBarName is the name of the SpeedBar to display.

To hide the standard SpeedBar, use the following command:

{Application.Display.Show_SpeedBar "No"}

Using Test mode

If you need to exit Test mode without running any link commands, choose Stop from the dialog box's Control menu. Clicking an OK or Cancel button also exits Test mode; if you've added link commands to the dialog box that run when OK or Cancel is clicked, those link commands will run.

Pressing *Esc* is the same as clicking Cancel.

Working with dialog boxes

You can use the icons on the Graphs page to create, rename, or copy dialog boxes, as described in the next table.

Table 6.1
Editing a dialog box using its icon

Right-clicking the icon lets you set various dialog box properties.

Operation	**Step(s)**
Edit	Double-click the dialog box icon.
Rename	Right-click the dialog box icon, choose Name, and enter the new name.
Delete	Click the dialog box icon and choose Edit | Clear (or press *Del*).
Copy to another notebook	Click the dialog box icon, choose Edit | Copy, select the destination notebook's Graphs page, and choose Edit | Paste. If both the source and the destination notebook appear on the Quattro Pro desktop, you can copy the dialog box by selecting the source notebook's Graphs page and dragging the dialog box icon into the destination notebook window.
Duplicate	Click the dialog box icon, choose Edit | Copy, then choose Edit | Paste. A duplicate can't have the same name as the original, so a unique name is automatically generated.
Move to another notebook	Select the dialog box icon, choose Edit | Cut to select the destination notebook's Graphs page, then choose Edit | Paste.

Working with controls

The information in this section applies, generally, to all controls, regardless whether the control is a radio button or a scroll bar.

Providing hints

When a user activates a control, a brief description of the control appears on the status line. To create this hint, right-click the control, choose Help Line, type the hint, and choose OK.

Control properties

For explanations of a control's properties (the settings you see when you right-click a control), choose Help | Contents | Application Building | Dialog Window Objects. From the list, choose the control you're interested in. A list of properties appears, with descriptions.

Selecting several controls

You can select more than one control at a time, to move them as a group:

1. Choose the Selection tool from the dialog window SpeedBar.
2. Hold down *Shift* and click each control you want to select. Handles surround each control as you select it.

To deselect a control, click it again.

Another way to select multiple controls is to choose the Selection tool and drag a selection frame around the controls.

To deselect a group of controls, choose the Selection tool and click the dialog window background.

Copying and moving controls

To move a control to a different dialog box (or SpeedBar), select the control, choose Edit | Cut to move it to the Clipboard, then choose Edit | Paste to copy it into another dialog window in another notebook. (The target notebook and dialog box must be open and active.) Each time you choose Edit | Paste a new copy of the control is created.

Caution! Deselect any controls in the active dialog window before using Edit | Paste, so the selected control is not changed (see page 331 for details).

To copy a control to the Clipboard without removing it from the active dialog window, select it and choose Edit | Copy.

Copying parent controls

Copying a parent control to the Clipboard copies any controls attached to it as well; removing the parent control from the dialog window also removes the attached controls. See page 335 for details on grouped controls.

Moving and sizing controls

To resize or move a control,

1. Choose the Selection tool.
2. Select the control by clicking it.
3. To move a control, point to it and drag it. To resize a control, drag one of its handles.

   If multiple controls are selected, dragging one control moves all the controls; dragging the handle of one control also resizes the other selected controls.

Selecting multiple controls (by holding down *Shift* while clicking them) and choosing Dialog | Align | Resize to Same resizes the selected controls to the same size as the first control selected.

Using the Dimension options

To specify a precise size and position for a control, right-click the control and choose Dimension.

Table 6.2 Dimension options for controls

Option	Description
X Pos	Distance in pixels between the left edge of the control and the left side of the dialog box
Y Pos	Distance in pixels between the top edge of the control and the bottom edge of the dialog box's title bar
Width	Width of the control, in pixels
Height	Height of the control, in pixels

Moving overlapping controls

The next table lists commands on the Dialog | Order menu that let you display a control that's beneath another control.

Table 6.3
Order menu commands to move overlapping controls

Command	**Description**
Bring Forward	Brings the selected control one step forward.
Send Backward	Sends the selected control one step backward.
Bring to Front	Places the selected control over any controls that covered it previously.
Send to Back	Places the selected control beneath any controls it covered previously.

Aligning controls

Right-clicking the dialog window background and choosing Grid Options provides three settings that help align controls.

Table 6.4
Table alignment settings for controls

Setting	**Description**
Grid Size	Lets you specify the distance in pixels between grid points.
Show Grid	Makes the grid visible. Dots appear in the back- ground to represent each grid point. (User's won't see this grid.)
Snap to Grid	Enables a hidden grid. Controls will align with the grid as you move them.

Aligning multiple controls

You can use commands on the Dialog | Align menu to align or reorder multiple controls.

Table 6.5
Align menu commands for multiple controls

Command	**Description**
Left	Aligns controls along the left edge of the leftmost object selected.
Right	Aligns controls along the right edge of the rightmost object selected.
Horizontal Center	Centers controls between the top and bottom of the dialog box.

Table 6.5: Align menu commands for multiple controls (continued)

Command	Description
Top	Aligns controls along the top edge of the topmost object selected.
Bottom	Aligns controls along the bottom edge of the bottommost object selected.
Vertical Center	Centers controls between the left and right sides of the dialog box.
Horizontal Space	Horizontally spaces controls by the amount of pixels you specify (the vertical position of the controls remains the same). It also reorders the controls based on the order in which you selected them (discussed next).
Vertical Space	Vertically spaces controls by the amount of pixels you specify (the horizontal position of the controls remain the same). It also reorders the controls based on the order in which you selected them (discussed next).
Resize to Same	Sets all selected controls to the same size as the first control selected.

Repositioning controls

The commands Horizontal Space and Vertical Space let you evenly space and arrange controls in a dialog box.

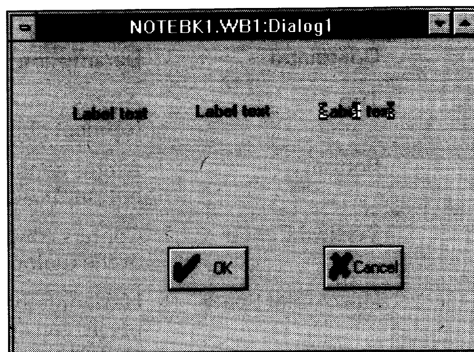
The following example shows how to reposition controls using the Horizontal Space command.

1. Choose Tools | UI Builder to display a new dialog window.

2. Create three labels (left to right) using the Label tool.

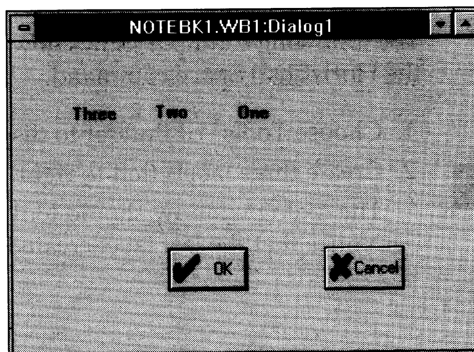
   The dialog window should look similar to the next figure. Notice that the last label created is selected.

Figure 6.21
The dialog box with three controls



3. Edit the label text to read `One`, `Two`, and `Three`, respectively.
4. Click the dialog window background to deselect the third label.
5. To reverse the order of these controls, hold down the *Shift* key while you click label Three, then label Two, then label One.
6. Choose Dialog | Align | Horizontal Space. A default spacing of 5 is already entered into the dialog box, so choose OK.
7. Click the dialog window background to deselect the labels. Notice that the labels moved according to the order in which you *selected* them (as shown in the next figure). The labels are also spaced evenly, five pixels apart.

Figure 6.22
The dialog box after reversing the order of the controls



Vertical Space behaves like Horizontal Space, but the first control you select moves to the top of the order, the second control moves just below the first, and so on.

Setting a control's value

Each control stores one *setting*; the user manipulates the control (by clicking a scroll arrow, typing text, and so on) to change that setting.

There are several ways to set a control's value:

- through initial settings you pass using a {DODIALOG} command
- by a user changing a setting from within the dialog box
- through the {SETOBJECTPROPERTY} command
- through link commands from any dialog control (see page 352).

For example, the following macro command uses the Value property to enter `Spanish` into an edit field,

{SETOBJECTPROPERTY "Dialog1:EditField3.Value","Spanish"}

Setting a control's initial value

The macro command {DODIALOG} displays the specified dialog box.

The next figure shows the macro that displays the dialog box shown in Figure 6.1 and uses the settings within the dialog box to calculate a loan payment. Cell D7 contains the {DODIALOG} command.

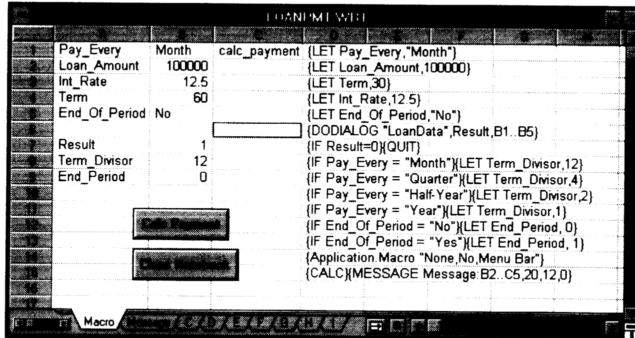


Figure 6.23
A macro to calculate loan payments

You can open LOANPMT.WB1 to study this example more closely.

The {DODIALOG} command in this figure is

{DODIALOG "LoanData", Result, B1..B5}

This command displays the LoanData dialog box, places the result of how the user closed the dialog box in the cell named Result,

and passes the values in cells B1 through B5 to the controls in the dialog box, in order.

If a control's Process Value property is set to No, it won't receive an initial value from the block specified in the {DODIALOG} command.

For example, if your dialog box has six controls, but you've set the Process Value property of the third control to No, your setting block should be only five cells long. The values in the first two cells will be passed to the first two controls. However, the value in the third cell will go to the fourth control, and the values in the fourth and fifth cells will go to the fifth and sixth controls, respectively.

Controls that don't have a Process Value property will receive no value from a {DODIALOG} command.

Changing the order in which you pass initial values

By default, the order in which you create the controls is the order in which you must pass each control's value (through the {DODIALOG} command). In other words, the first cell in the specified block sends its value to the first control you created *no matter where it is positioned within the dialog box.*

For example, the last argument in the command {DODIALOG "LoanData", Result, B1..B5} specifies that cells B1 through B5 contain initial values for the controls in the dialog box. Cell B1 must contain the initial value for the first control you created, cell B2 must contain the value for the second one, and so on.

Page 329 describes the other ways that a control's value can be assigned.

To change the order of the controls in the dialog box to match the order of the values in the setting block,

1. Click the dialog window background to deselect its controls.
2. Hold down the *Shift* key and, in the order in which you want to pass initial settings, click each control.
3. Choose Dialog | Order | Order Controls.

   The first control you clicked will receive its initial value from the first cell of the block, the second control you click will receive its initial value from the second cell of the block, and so on, regardless of their arrangement within the dialog box.

Editing the setting order

If you have many controls in your dialog box, you can use a shortcut to re-order the controls to match the order of the values in the setting block. Use Dialog | Order | Order From to place related controls together in the setting order.

For example, suppose you have six controls in this order:

1 2 3 4 5 6

But the setting block in your notebook already has the values in this order:

1 2 5 4 3 6

The only controls that are out of order are 4 and 3; you want them to follow 2. You can reorder the controls to match this order:

1. Click the dialog window background to deselect its controls.
2. Hold down the *Shift* key, then click control 2, then control 3, then control 4.
3. Choose Dialog | Order | Order From.

Changing a control's text

If a control has a Label Text property, which is set to the title appearing on the control, you can set the property by double-clicking the control, typing the new text (or editing the existing text) and pressing *Enter*.

Using the Clipboard

When the Clipboard contains text from Quattro Pro or another Windows application, you can paste the text into these controls:

Table 6.6
Pasting text into a control

Control	Effect
Buttons	The text becomes the button title.
List boxes, Combo boxes, Pick Lists	The text becomes the list; each line becomes one item. The first item becomes the initial title.
Group boxes	Creates radio buttons in the group box; each line becomes the title of one radio button.
Edit fields	The text becomes its new value.

If the Clipboard contains a bitmap, you can select a bitmap button and choose Edit | Paste to paste the bitmap onto the button's face. This bitmap is saved with the dialog box.

You can also select a dialog control and choose Edit | Copy to copy control titles, lists, or bitmaps to the Clipboard to paste them into other areas of Quattro Pro and other Windows applications.

Assigning a shortcut key to a control

An underlined letter in the text of a control acts as a shortcut key; it indicates that the user can press *Alt* and that letter to activate the control. To specify this underlined letter,

1. Double-click the control's text.
2. Move the insertion point to just *before* the letter to underline.
3. Type an ampersand (&). For example, B&utton sets the title to Button.
4. Press *Enter*.

You can also assign a shortcut letter to a control that has no title (such as an edit field or a list box):

1. Right-click the control.
2. Choose Name.
3. Move the insertion point to just *before* the letter to use. You can type a new name for the control if needed.
4. Type an ampersand (&). Then press *Enter*.

When a control doesn't have a title, you can place a shortcut key in the control's label. Just double-click the label and type an ampersand to the left of the letter you want to use as the shortcut. Press *Enter* to save the new label text. Use only one ampersand character in the label; if you add more than one, the first is recognized as a shortcut key designation, the remaining ones will appear in the label as text.

You can add alternative shortcut keys to controls using link commands. See page 354 for details on the event `key`, which you can use to run a link command when a specific key is pressed.

Setting tab order

A user can press *Tab* to cycle through each control in a dialog box. You can disable this feature for a particular control by right-clicking the control and setting the Tab Stop property to No.

To specify the order in which *Tab* moves from control to control, hold down the *Shift* key and select each control in the order you want them to cycle. Then choose Dialog | Order | Order Tab Controls to save the new order.

You can pull specific controls out of the tab order and group them together using Dialog | Order | Order Tabs From. For example, suppose the following numbers represented the existing tab order:

1 2 3 4 5 6

If you select items 2, 4 and 6, choosing Order Tabs From would change the tab order as follows:

1 2 4 6 3 5

Selection order within a group box

You can press Space to check the active check box button.

When check box buttons are placed in a group box, the user can activate one of the check boxes and then use *Tab* or arrow keys to select the next check box. To change the order in which the check boxes are selected,

1. Choose the Selection tool from the dialog window SpeedBar.
2. Click the dialog window's background to deselect any selected controls.
3. Hold down *Shift* and then click each check box, in the order in which you want the user to scroll through them.
4. Drag the check boxes until the pointer changes to a small hand. This saves the new order.

When radio buttons are placed in a group box, the user can activate one of the radio buttons and then use arrow keys to select and choose the next radio button. You can set this selection order in the same way as check boxes: Select the radio buttons in the order you want the user to scroll through them and then move them slightly to save the new order.

Disabling and concealing controls

Dialog controls have a variety of properties that you can use to temporarily hide or disable them. You can use these properties in conjunction with link commands to hide unneeded controls or reveal special settings.

Table 6.7
Properties to hide or disable dialog controls

Property	Purpose
Grayed	Specifies whether the control can be used. When set to Yes, the control appears dimmer to indicate it is unavailable.
Disabled	Specifies whether the control can be used, like Grayed, but doesn't dim the control when it's set to Yes.
Depend On	Specifies the areas of Quattro Pro in which the control is enabled (discussed next).
Enabled	The opposite of Disabled (setting this property to No disables the control). It's used by link commands and doesn't appear in a control's Object Inspector.
Hidden	Setting this property to Yes hides the control when the dialog box displays. (The control still appears in the dialog window while you're editing.)
Show	The opposite of Hidden (setting this property to No hides the control). It's used by link commands and doesn't appear in the control Object Inspector.

The most useful property for disabling controls is Depend On. Use it to specify the areas of Quattro Pro in which the control is disabled. For example, some controls are appropriate only when a graph window is active. To create a control that's available only when a graph window is active,

1. Right-click the control and choose Depend On.
2. Uncheck all check boxes except Graph to specify that this control is available only when a graph window is active.

Each check box button controls one area of Quattro Pro, as shown in the next table.

Table 6.8
Quattro Pro areas

Area	When its check box is checked
Desktop	Enables the control when no windows are open.
Notebook	Enables the control when a notebook window is active.
Graph	Enables the control when a graph window is active.
Dialog	Enables the control when a dialog window is active.
Input Line	Enables the control when the input line is active (when typing a new entry or editing a cell).
Graphs Page	Enables the control when a notebook window is active and the Graphs Page is selected.

If all options are checked, the dialog control is always available.

Grouping controls

Some dialog controls have an Attach Child property that makes controls dragged on top of them become *attached*. The control on top is then a *child* of the control beneath it, which is the *parent*. Grouping controls provides many advantages to the developer:

- Moving the parent control moves any attached elements.
- Copying the parent control copies any attached elements.
- Hiding the parent control hides any attached elements.
- Disabling the parent control disables any attached elements.
- Resizing the parent control resizes any attached elements proportionally (also see the discussion of the Position Adjust property on page 336).

Child controls must cover the parent control only; controls partially covering one another won't attach. Here's an example of grouping controls together:

1. Choose Tools | UI Builder to display a new dialog window.
2. Use the Rectangle tool to create a rectangle that's big enough to contain the OK button.
3. Move the OK button into the rectangle. The OK button is now attached to the rectangle.
4. Move the rectangle; the OK button stays with the rectangle.
5. Right-click the rectangle, choose Disabled and set it to Yes.
6. Choose the Test button to test the dialog box's operation.

7. Click the OK button; nothing happens, since the rectangle (its parent control) is disabled.
8. Click the Cancel button to exit Test mode and return to the dialog window.

To prevent an object from becoming a parent control, right-click it and set the Attach Child property to No. This doesn't affect controls already attached.

All controls in a dialog window are child controls of the dialog box (or SpeedBar), which is the parent control. Therefore you can disable all controls in a dialog box by right-clicking the dialog box and setting Disabled to Yes.

Nesting grouped controls

You can nest attached controls (for example, you can move a parent control on top of another control to attach them). The attached controls still stay associated with the original parent control, not that control's new parent.

Resizing grouped controls

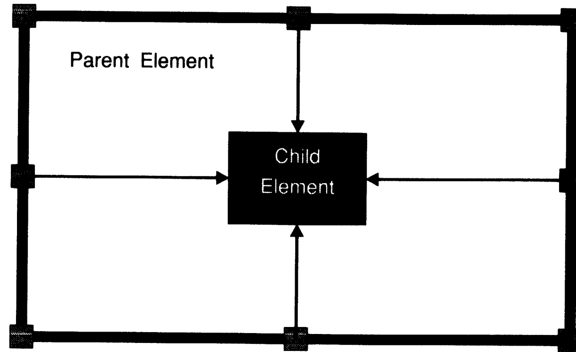
Moving a parent control moves any child controls with it. (Moving child controls doesn't affect the parent.) You can use a child control's Position Adjust property to specify how it moves (or resizes) when the parent control containing it is resized.

By default, a child control moves when a handle on the left or top side of its parent control is dragged. If you right-click a child control, then choose Position Adjust, and check Depend on parent, each edge of the child control is locked into place so that when the parent control is resized, the child control resizes proportionally.

When a handle of the parent control is dragged while Position Adjust is enabled, an edge of the child control is pushed or pulled, as shown by the arrows in the next figure.

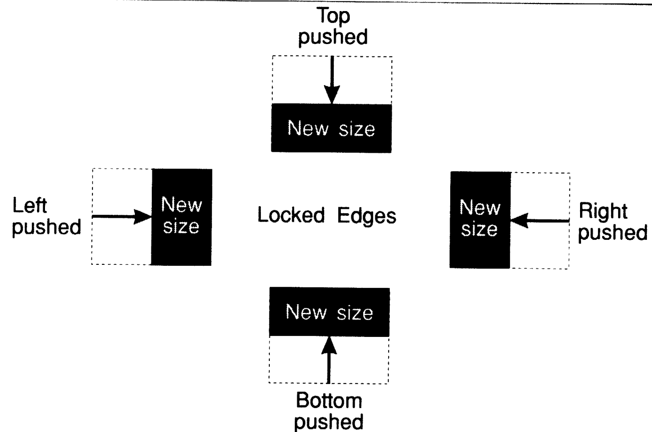
Figure 6.24
How dragging a handle of a parent control affects the child control

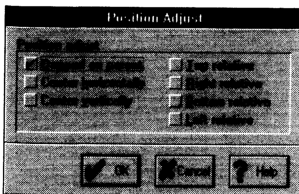
Dragging a corner handle affects two sides of the child control. For example, dragging the lower right handle would affect the bottom and right edges of the child control.



If the edge of a child control is pushed toward or pulled away from an edge of the child control that's locked, the child control is resized, as shown in the next figure. Arrows point to the edge being pushed.

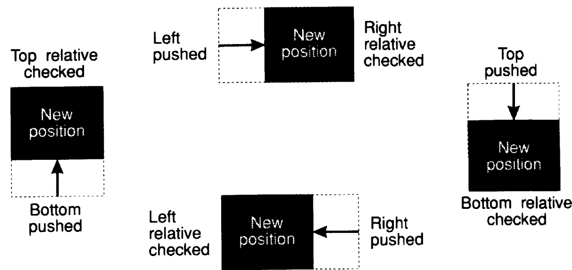
Figure 6.25
How resizing a parent affects child controls





You can use the check boxes in the Position Adjust dialog box to unlock edges of the child control. If an edge of the child control is pushed toward or pulled away from an unlocked edge, the child control moves instead of resizing, as shown in the next figure. Arrows point to the edge being pushed.

Figure 6.26
How child controls move when an edge is unlocked



You can check Center Horizontally to center the control between the left and right sides of the parent control whenever the parent control is resized. Check Center Vertically to center the control between the top and bottom edges of the parent control.

This example shows how to use Position Adjust:

1. Choose Tools | UI Builder to display a new dialog window.
2. The OK and Cancel buttons are child controls of the dialog box. Right-click the OK button and choose Position Adjust. Check Depend on parent to enable automatic positioning. Choose OK to save the setting.
3. Repeat the previous step for the Cancel button.
4. Drag the lower right corner of the dialog window to shrink the dialog window. Notice that both buttons shrink proportionally.

To make the Cancel button move instead of resizing,

1. Right-click the Cancel button and choose Position Adjust.
2. Check Top Relative and Left Relative to unlock the top and left edges of the Cancel button.
3. Choose OK to save the change and drag the lower right corner of the dialog window to shrink it.

Notice that the OK Button still shrinks, but the Cancel button moves. Now whenever the right or bottom edge of the dialog box is dragged, the Cancel button moves instead of shrinking.

Connecting controls to cells or other controls

The Dialog | Connect command is a quick way to link a control to another control or to a notebook cell. This command exchanges settings by creating a link command in the selected object (link commands are discussed on page 352). Use Dialog | Links to modify "connected" link commands.

To connect a control:

1. Select the control you want to connect.
2. Choose Dialog | Connect, then select the cell (or control) you want to link to. The linked object's name or address appears in the Target edit field; the control's name appears in Source.
3. If you want to send changes in the control's value to the target cell or control immediately, check Dynamic Connection; otherwise, the target object will be updated only when the dialog box is closed.
4. Choose OK to save the connection and return to the dialog window.

This example connects a scroll bar to a notebook cell:

1. Select cell A1 of the notebook and enter `16`. This will be the initial value for the scroll bar.
2. Choose Tools | UI Builder.
3. Create a vertical scroll bar.

4. Choose Dialog | Connect. Click cell A1, then choose OK.
5. To test the link, choose the Test button. The scroll box on the scroll bar will jump to a new position after receiving its initial value from cell A1.

6. Drag the scroll box; the value stored in cell A1 changes.
7. Choose OK to exit Test mode.

Note To delete a "connected" link, choose Dialog | Links, select the correct link, then click Delete.

Types of controls

This section gives information specific to each control type.

To change some aspect of a control, right-click it to reach its Object Inspector. The Object Inspector lists properties specific to that type of control.

Common button properties

The properties in the next table are common to push buttons, check boxes, radio buttons, and bitmap buttons.

Table 6.9: Properties common to push buttons, radio buttons, and check boxes

Property	**Description**
Draw to Right	Determines whether the button appears to the right of the title (Yes) or to the left (No).
Label Text	Contains the text appearing on the push button (or bitmap button), next to the check box button, or next to the radio button. To follow Windows conventions, when a push button leads to another dialog box, you should end its text with an ellipsis (...).
Text Draw Flags	Determines where the text appears in relation to its background (for check boxes and radio buttons) or in relation to the button itself (for push buttons and bitmap buttons). This property has the same options as a label's Text Draw Flags property (see page 342).
Button Type	Specifies the button's behavior. For example, to change a push button into a check box button, set Button Type to Check Box. Two special button types (OK Exit Button and Cancel Exit Button) change the button into a push button that closes the dialog box (see page 342).
Default Button	Specifies whether this button is automatically clicked when the user presses *Enter* in a dialog box or SpeedBar. A button configured this way is called the *default button*. A button defined as the default button appears in the dialog box, with a thick black border. Only one push button in the dialog box can have this property set to Yes; by default, it's the OK button.

Push buttons

A push button runs a specified command when the user clicks it.

To have a macro run when the user clicks a push button, add a DOMACRO link command to the push button. See page 355 for an example.

Push buttons don't have a value.

To run a macro that is stored in the notebook when the user clicks the push button, add a DOMACRO link command that contains a {BRANCH}.

To make a push button close the dialog box, use the Button Type property. See Table 6.10 for details.

Check boxes

Check boxes present a Yes/No option to the user. The text that appears in this button is stored in its Label Text property. The value of a check box is Yes when it's checked, No when it's unchecked.

Radio buttons

Radio buttons, when placed in a group box with other radio buttons, present a list of mutually exclusive choices to the user—only one radio button can be chosen at a time. When the user chooses one of the radio buttons, it darkens to indicate that it's chosen.

To automatically fill a group box with radio buttons, paste a block stored in the Clipboard into a group box; one radio button is created for each row of the block.

There is a shortcut for this: hold down *Ctrl* and drag the lower right handle of the group box to increase its size. When you release the handle, radio buttons appear within. Repeat this process until the desired number of radio buttons appear.

By default, a radio button's Process Value property is set to No, and the group box containing it has Process Value set to Yes.

When its Process Value property is set to Yes, a radio button has a value of Yes when chosen, No when not chosen.

Bitmap buttons

A bitmap button has a graphic image on its face instead of text. It can imitate the behavior of a check box, radio button, or push button.

There are several different ways to specify the bitmap that appears on a bitmap button's face:

- Right-click the bitmap button and choose Bitmap. You'll see a list of predefined bitmaps. Select a bitmap from the list (it then appears under Preview) and choose OK.
- Choose Browse from the Bitmap property dialog box, specify the name of an external Windows bitmap (BMP) file, and choose OK. The bitmap will be saved with the active notebook.
- If the Clipboard contains a bitmap, select the bitmap button and choose Edit | Paste to paste the bitmap onto the button's face. The bitmap will be saved with the active notebook.

Bitmap buttons have the value of whatever button they're imitating.

To specify how the button behaves, set the bitmap button's Button Type property.

Table 6.10
Types of bitmap buttons

Button type	Description
Push button	Acts like a push button. You can use link commands to make it perform operations when a user clicks it.
Radio button	Acts like a radio button, including a radio button's behavior in the group box (see page 341). It appears recessed when a user chooses it.
Check box	Acts like a check box button. It appears recessed when a user checks it.
OK Exit button	Closes the dialog box like a standard Windows OK button. If the dialog box was displayed by {DODIALOG}, the control values are written back to the setting block.
Cancel Exit button	Closes the dialog box like a standard Windows Cancel button. If the dialog box was displayed by {DODIALOG}, the control values aren't written back to the setting block, since the dialog box was canceled.

Labels

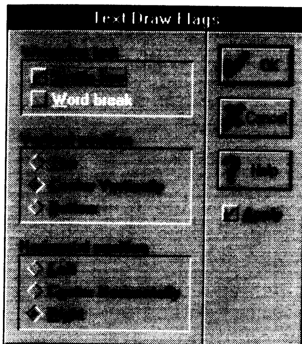
A label is simply descriptive text that you place beside another control to describe the control's purpose to the user.

Labels don't have a value.

Resizing a label doesn't increase the text size; it makes a larger background to display the text on.

To see the background of a label, select it; By default, the label text appears in the upper left corner of its background; to specify a different position for the label text, right-click the label and use the Text Draw Flags property. You can also use Text Draw Flags to wordwrap the label text so that it fits in the label background.

Table 6.11
Text draw options



Option	Description
Apply	Applies the Text Draw Flags properties you've selected.
Wrapping Text	**Single Line** specifies whether the label converts formatting codes. Uncheck this to allow formatting codes to be entered into the label (discussed next). This also enables the Word Break option. **Word Break** specifies whether to wrap the text onto a new line in the label background. Check this to enable line wrapping (available only when Single Line is unchecked).
Vertical Position	**Top** places text flush against the top edge of the label background. **Center Vertically** centers the text between the top and bottom of the background. It doesn't affect the horizontal position of the text. **Bottom** places text flush against the bottom of the background.
Horizontal Position	**Left** places text flush against the left side of the label background. **Center Horizontally** centers the text between the left and right sides of the label background. It doesn't affect the vertical position of the text. **Right** places text flush against the right side of the label background.

You can enter two special text items in a label to change its appearance:

- When the Single Line option (in the Text Draw Flags property) is unchecked, you can enter the following *escape sequences*:
  - **\n** (or **\r**) makes characters to the right of it display on a second line. This only works when the Single Line option (in the Text Draw Flags property) is unchecked.
  - **\t** inserts a tab into the text (a fixed amount of space that's handy for lining up columns of text). This only works when the Single Line option (in the Text Draw Flags property) is unchecked.

- \\ inserts a backslash (\) into the text. You can use this to enter \n, \r, or \t into a label as normal text.

- **&** makes characters to the right of it appear underlined in the label text (the ampersand doesn't appear). This is handy for naming controls.

Edit fields

An edit field is where a user types specific information.

You can use the Field Type property to restrict the type of information an edit field accepts.

Table 6.12
Restricting information in an edit field

Field type	Result
Integer	Restricts user entry to integers. This adds three properties to the edit field Object Inspector (which is renamed Edit Integer). **Minimum** defines the lowest acceptable value users can enter. **Maximum** defines the highest acceptable value a user can enter. **Default** defines the initial value of the edit field.
String	Accepts any text a user enters (including numbers); this is the default edit field type.
Real	Accepts any number or formula.
Block	Accepts only cell addresses or block coordinates.
Hidden	Accepts any text, but displays a pound sign (#) for each character instead of the actual character. This is handy when users are entering passwords.

The value of an edit field is the text entered into it.

Edit fields have additional properties you can set to customize their appearance and behavior.

Table 6.13
Additional edit field properties

Property	Description
Allow Point Mode	When set to Yes, lets a user point to cells and blocks in the notebook (all Field Types except Integer).
Show Frame	When set to No, removes the default frame from the edit field.
Edit Length	Sets the maximum number of characters a user can type into the edit field.

Table 6.13: Additional edit field properties (continued)

Property	Description
Terminate Dialog	When set to No, keeps the dialog box from activating its default button when the user presses *Enter* in an edit field. (See page 340 for information on the default button.) This is important in SpeedBars, since activating the default button would close the SpeedBar.
Convert Text	Lets the user enter escape sequences into the edit field: Entering **\t** inserts a tab character for lining up columns of text within the edit field; entering **\n** (or **\r**) places the remaining text on a new line. \\ Inserts a backslash (\) into the text.

Spin controls

A spin control is an edit field with two arrows on its edge. The user can either click an arrow to increase or decrease the spin control's value, or type the value.

The value of a spin control is always an integer. (You can use edit fields to enter decimal numbers.) Three properties control the values a spin control will accept, as shown in the following table.

Table 6.14
Restricting spin control values

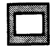
Property	Description
Minimum	Smallest integer that the control will accept
Maximum	Largest integer that the control will accept
Default	The initial value of the control when the dialog box displays

Spin controls have additional properties to customize their appearance and behavior.

Table 6.15
Additional spin control properties

Property	Description
Show Frame	When set to No, removes the default frame from the spin control.
Edit Length	Sets the maximum number of characters a user can type into the spin control.

Rectangles



A rectangle can be used to visually organize controls in a dialog box.

Rectangles don't have a value.

To change how a rectangle appears, right-click it and choose Rectangle Style.

Table 6.16
Rectangle Style property options

Option	Description
Plain	Draws the rectangle as a solid box without a frame.
Framed	Draws a frame around the rectangle (the default). To set the color of this frame, use the Frame Color property.
Beveled Out	Displays a slightly raised rectangle.
Beveled In	Displays a slightly recessed rectangle.
Transparent	Displays a clear, unfilled rectangle. This is handy for grouping controls together without displaying the rectangle they're attached to. See page 335 for details on grouping controls.

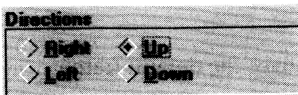
To change a rectangle's color, right-click the rectangle, then choose Fill Color.

Group boxes

A group box usually contains other controls such as radio buttons or check boxes. It looks like a rectangle with a title at the top.

When a group box contains radio buttons, its value is the title of the radio button chosen (see page 341); otherwise, it does not have a value.

Initially, the Process Value of a group box is set to Yes, and the Process Values of radio buttons within the group box are set to No.



In addition to the Value property, the group box has a property called Selected that indicates numerically which of its radio buttons is chosen, starting from the top left radio button (0) and proceeding to the bottom right button. For example, the group box shown at left has Selected set to 1. If Right was chosen, Selected would be 0. Left and Down would set Selected to 2 and 3, respectively.

List boxes

A list box presents the user with a list of items to choose from. Whichever item the user picks becomes the value of the list box.

There are two ways to specify the items that appear in a list box:

- **Using the Clipboard**. Copy a block to the Clipboard, then select the list box and choose Edit | Paste. Each row of the block becomes one item in the list.
- **Using the List property**. If your list is already in a block in the notebook, right-click the list box, choose List, type or point to the block coordinates, and choose OK.

List box properties change the way items appear.

Table 6.17
List box display properties

Property	Description
Number Of Columns	Specifies how many columns of items appear in the list box. The default is one column.
Ordered	Sorts items in ascending order when set to Yes. (Resetting Ordered back to No does *not* restore the original order.)

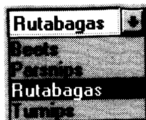
The list box property Selected indicates numerically which of its items is chosen, starting from the top item (0) and proceeding row by row to the bottom right. For example, if the first item was chosen, Selected is set to 0; the fifth item would set Selected to 4. The text of the item chosen is stored in the Selection Text property.

Combo boxes

Combo boxes (also called *drop-down list boxes*) are a special type of list box that lets the user add items to the list. They consist of an edit field, a down arrow button, and a list box (which appears only when the user clicks the down arrow or presses ↓ in the edit field).

The value of the combo box is the value of its edit field. There are several ways users can change a combo box's value:

- They can choose the edit field and enter the value. (You can disable this by setting No Edit to Yes.)



- They can press the ↓ key to move through the list, then press *Enter* on the desired choice. (Pressing *Enter* also clicks the dialog box's default button, which typically closes the dialog box. To disable this, set Terminate Dialog to No.)
- They can click the down arrow button to display the list, then click the desired choice. (To remove a combo box's down arrow button, set Add Down Button to No.)

Combo boxes have an Ordered and List property identical to those properties in a list box. You can also use the Clipboard to specify the list that a combo box displays. (See the previous section.)

You can make a combo box store a history list of the entries that have been typed into its edit field. This history displays as the combo box's list. See page 368 for an example.

Pick lists

A pick list button displays a list of choices when the user points to it and holds down the left mouse button. If the list is empty, the user can double-click the pick list button and type a setting.

The value of a pick list is whichever item the user selects (the value is initially set to the first item in the pick list).

To create a pick list,

1. Enter the items of the pick list as labels in a block in the notebook, one label per row.
2. Select the block and choose Edit | Copy.
3. Activate the dialog window you want to add the pick list to.
4. Choose the Pick List tool from the dialog SpeedBar and click in the dialog window to create the pick list button.

5. Choose Edit | Paste to create the list. The first item in the list displays on the pick list button's face.
6. Use the Title property, if desired, to specify a title to display initially on the pick list button. This title doesn't have to be an item on the list.

You can use a link command to set the Title property whenever the dialog box displays.

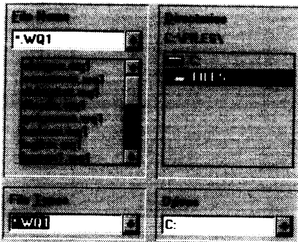
Choosing an item from the list changes the value of the list. By default, a pick list button doesn't resize when a user chooses an item too large to display on the button's face. You can set Resize to

Yes to make it expand or shrink depending on the size of the item the user picks.

File controls

A file control button displays a list of files on disk. From this list, the user can choose a file name, directory, and drive.

The value of a file control determines the directory, drive letter, and file name (or wildcard) that appears. For example, the value `C:\FILES*.WB1` means

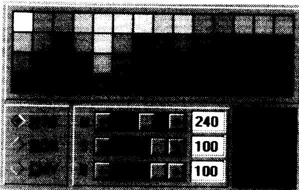


- Drives is set to C:.
- Directories is set to C:\FILES\.
- File Name is set to *.WB1.
- File Types is set to *.WB1 (if a wildcard is specified; otherwise File Types isn't affected by the value).

Whenever a user chooses a different directory to view or a different file name or wildcard to use, the change is reflected in the file control's value. See page 366 for an example of using an edit field to change a file control's value.

File controls are large; make sure the dialog window is large enough to accommodate a file control before you add one.

Color controls



A color control lets the user change a color setting. Its value is in the format *Red,Green,Blue* where *Red* is the amount of red in the color, *Green* the amount of green, and *Blue* is the amount of blue. Each amount is a number from 0 to 255. For example, black is 0,0,0; white is 255,255,255; and red is 255,0,0.

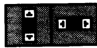
The current color setting appears in the large rectangle on the right side of the control.

The user can click HSB or CMY (on the color control) to specify the color using a different color model (see Chapter 9 of the *User's Guide* for more information on color models). The value of the color control doesn't change when a different color model is used to specify the color; the control value is always in Red,Green,Blue format.

Note When setting a color control using {DODIALOG}, *three* cells in the setting block are needed. The first contains the amount of red in the color, the second the amount of green, and the third the amount of blue. See page 330 for more information on the setting block.

Color controls are large; make sure the dialog window is large enough to accommodate a color control before you add one.

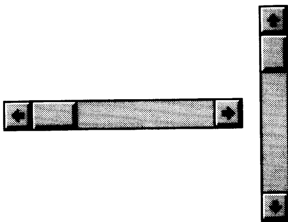
Scroll bars



Scroll bars let the user pick a value from a fixed range of values. You can use a vertical scroll bar or a horizontal one. The value of a scroll bar is an integer from a fixed range of integers.

The next table lists the options you can set for horizontal and vertical scroll bars to restrict the range of values they accept and to specify how manipulating the scroll bar changes its value.

Table 6.18
Scroll bar Parameter options (the Parameters property)

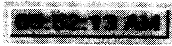


Option	Description
Min	A horizontal bar's value when its scroll box is as far left as possible; vertical bars are set to this value when their scroll box is at the top of the scroll bar.
Max	A horizontal bar's value when its scroll box is as far right as possible; vertical bars are set to this value when their scroll box is at the bottom of the scroll bar.
Line	The amount the scroll bar's value changes when a user clicks a scroll arrow.
Page	The amount the scroll bar's value changes when a user clicks above or below the scroll box on a vertical scroll bar, or to the left or right of the scroll box on a horizontal scroll bar.
Time	The amount of time in milliseconds that Quattro Pro waits before moving the scroll box in response to a user action (clicking a scroll bar, holding down the mouse button while pointing to a scroll arrow, and so on).

For a complete explanation of scroll bars, scroll boxes, and scroll arrows, see the Windows *User's Guide*.

Time controls

A time control shows the user the current time. You can use time controls to display the current time in a dialog box or SpeedBar. To add a *clock*,



1. Create a time control in the dialog window.
2. Right-click the time control and set Show Time to Yes. The timer changes into a small digital clock.
3. Right-click the timer and set Timer On to Yes. This enables the clock.

Time controls don't have a value.

You can use a time control to run link commands at regular intervals or at a certain time of day. This is handy for creating SpeedBar controls that periodically retrieve current values from the active window.

There are two ways to configure a time control for generating regular events: as a Timer and as an Alarm.

As a **Timer**, it generates the Timer event at regular intervals (for example, every two seconds). Two properties of the time control are used to specify the number of milliseconds to wait between Timer events, as shown in the following formula:

Wait = Units in Milliseconds × Interval in Units

Units in Milliseconds is set to 1000 by default, so you can use the Interval in Units property to specify how many seconds to wait between Timer events. To make a timer,

1. Right-click the time control, choose Units in Milliseconds and enter how many milliseconds it takes for one unit to elapse.
2. Right-click the time control, choose Interval in Units and enter how many units to wait before generating a Timer event.
3. Right-click the time control, choose Timer On and set it to Yes.

As an **Alarm**, the time control generates the Alarm event at a certain time of day (for example, at 5:00). The SpeedBar must be open; for example, an alarm set for 3:00 won't generate an Alarm event if the SpeedBar is not onscreen at 3:00. To make a time control an Alarm,

1. Right-click the time control, choose Alarm On and set it to Yes.
2. Right-click the time control, choose Alarm Time and enter the time that the Alarm event should be generated.

A time control can have both of these functions enabled simultaneously. For example, you could have a timer generate Timer events every two seconds and send out an alarm event at 5:00 to signal the end of the work day.

See page 364 for an example of using a time control to run link commands.

Working with Link commands

Link commands are statements you attach to a control to indicate what should happen when the user changes or selects the control.

A link command can do many things; it can send a value to a cell in the notebook; it can run a macro; it can close the dialog box; it can change the zoom factor in the notebook.

Link commands usually specify at least three things: an event, an action, and an object. The event indicates what occurrence should trigger the link command; the action indicates what should happen; the object is what should be acted upon.

One control can have many link commands.

The Object Link dialog box

To add link commands to a control, select the control and choose Dialog | Links.

The next figure shows the Object Links dialog box for a control that has two link commands assigned to it.

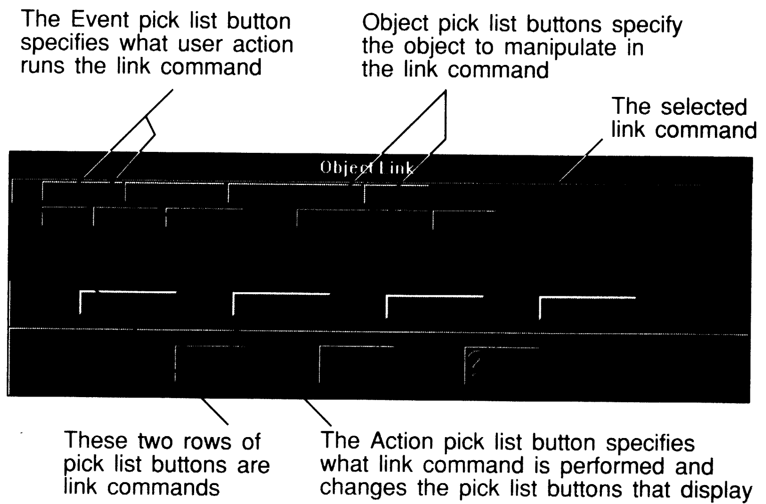


Figure 6.27
The Object Links dialog box

Each row of pick list buttons in this figure is a *link command*. The selected link command appears darker (in the figure, the second link command is selected).

- **Add** creates a new link command *below* the selected link command.
- **Insert** creates a new link command *above* the selected link command.
- **Delete** removes the selected link command from the control.
- **Delete All** removes all link commands from the control.

Link events

A link command runs in response to some *event* that occurs in a dialog box (or SpeedBar). The first pick list button shows what specific event you want to trigger the link command. In the previous figure, the event is `Clicked` for the first link command; `Init` for the second.

Note A control can have more than one link command respond to the same event. The order in which the link commands appear in the Object Link dialog box is the order they'll run in.

The next table lists each event and what it detects.

Table 6.19: Events that can initiate a link command

Event	Will trigger the link command when...
Init	The dialog box is about to display. Use this to set initial settings.
Init Complete	Available only when adding a link command to the dialog box itself, this event occurs immediately after all controls have run all link commands that respond to `Init`.
OKExit	The user closes the dialog box. The link command runs before the dialog box closes.
CancelExit	The user cancels the dialog box. The link command runs before the dialog box closes.
Clicked	The user clicks the control.
Right_bdown	The user right-clicks the control.
Left_bdown	The user points to the control while holding down the left mouse button. Releasing the button generates a `Clicked` event.
Doubleclick	The user double-clicks the control.
Valuechanged	The user changes the value of the control. This event occurs anytime the value is changed (by the user, by a link command, or by a macro command).
key:*keystroke*	The user presses the key *keystroke*. When you choose `key` from the event pick list, it flashes to indicate that you must press a key. Press the key you want to generate the event. The key displays after `key:` in the pick list button when it's set. For example, `key:Ctrl+F5` indicates that the link command is triggered by *Ctrl+F5*.
Activate	The user has chosen the control for manipulation. For example, clicking an edit field generates this event. Use `Valuechanged` to trap the final result.
Deactivate	The user has chosen another control, leaving this control inactive.
Lineup	The user increases the scroll bar's value by clicking a scroll arrow (available only on scroll bar controls).
Linedown	The user decreases the scroll bar's value by clicking a scroll arrow (available only on scroll bar controls).
Pageup	The user increases the scroll bar's value by clicking it between a scroll arrow and the scroll box (available only on scroll bar controls).
Pagedown	The user decreases the scroll bar's value by clicking it between a scroll arrow and the scroll box (available only on scroll bar controls).
Thumb	The user clicks or drags the scroll bar's scroll box (scroll bar controls only).
Editdynamic	The user is inserting or deleting characters in the combo box's edit field (combo box controls only).
Trigger	This event can't be generated by any user action; use it to set link commands that can be run only by other link commands. You can use the link command TRIGGER to generate this event.

Table 6.19: Events that can initiate a link command (continued)

Event	Will trigger the link command when...
Alarm	The time of day specified in the timer's Alarm Time property has been reached (time controls only).
Timer	The amount of time indicated in the timer's Timer Interval property has elapsed (time controls only).

Assigning a link command to a control

To add a link command to a control,

1. Select the control, then choose Dialog | Links.
2. Choose Add to create a new link command.
3. Point to the first pick list button (the *Event pick list*). Hold down the left mouse button, drag to the event that should initiate the link command, then release.
4. Point to the second button (the *Action pick list*). Hold down the mouse button, drag to the desired command, then release.
5. The remaining pick list buttons change, depending on which action you chose in step 4. Set the remaining buttons as needed.
6. Repeat steps 2 through 5 to create additional link commands.
7. Choose OK to save the link commands and return to the dialog window.

Link command examples

This section contains examples of each type of link command.

Example 1: A link command that runs a macro

This example shows how to create a link command that generates a warning beep when the user cancels the dialog box.

1. Choose Tools | UI Builder to display a new dialog window.
2. Choose Dialog | Links. (When no controls are selected, Dialog | Links adds link commands the dialog box.) Choose Add to create a new link command.
3. Point to the event pick list button (Init appears on it) and hold down the left mouse button. Drag the pointer to CancelExit and release the mouse button. `CancelExit` appears on the pick list button.

This specifies that the link command will run when the user cancels the dialog box.

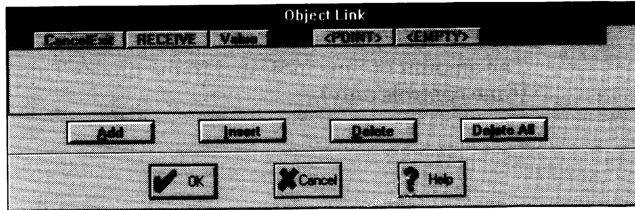


Figure 6.28
The Object Link dialog box after specifying CancelExit as the event

4. Next, specify what action should occur when the user cancels the dialog box: the link command should run a macro.

   The command {BEEP} sounds the computer's speaker.

   To make the computer beep, point to the action pick list button (RECEIVE appears on it), hold down the left mouse button, drag the pointer to DOMACRO and release. DOMACRO appears on the pick list button.

   Notice that the remaining pick list buttons are replaced by an edit field.

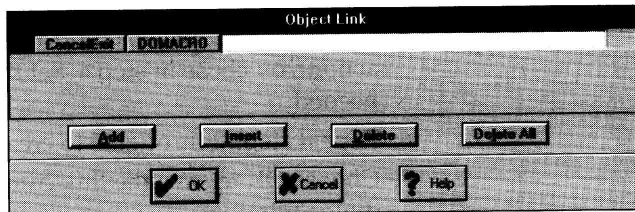


Figure 6.29
The Object Link dialog box after specifying DOMACRO as the action

5. Click the edit field, then enter {BEEP}{BEEP}{BEEP}. This specifies that the link command should beep three times when the user cancels the dialog box.

   The Links dialog box should look like the next figure.

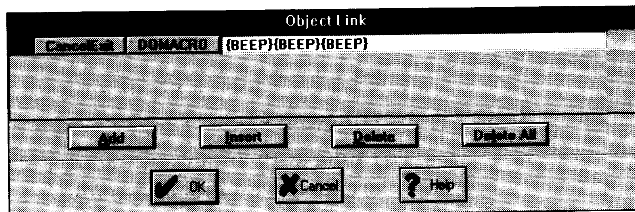


Figure 6.30
The Object Link dialog box after specifying the macro

6. Choose OK to save the link command and return to the dialog window.

7. Choose the Test button from the SpeedBar to test the dialog box's operation.
8. Click Cancel. The computer beeps before exiting Test mode.

Example 2: A link command that changes a notebook's zoom factor

There are two link commands that change properties in another object: SET and SEND. The object to change is specified by the *object pick list* button. The object pick list contains the following choices:

Table 6.20 Items in the object pick list

Command	**What it does...**
<POINT>	Lets you click (point to) the object you want to affect. You can point to a dialog box, dialog controls, or notebook blocks. The object's name (see Appendix B) will appear on the pick list button.
<ENTER>	Lets you type text that identifies the object, using the same syntax you use to identify an object in @PROPERTY or a macro command.
Active_Notebook	Changes a property of the active notebook.
Active_Page	Changes a property of the active page.
Active_Block	Changes a property of the active block.
Active_Object	Changes a property of a selected object. You must double-click the pick list button to the right of the object pick list button, and type the name of a property to use. (See Appendix B for a list of properties.)
Application	Changes properties of the application.

In some situations, <EMPTY> appears on the pick list button to the right of the object pick list button. <EMPTY> indicates that the property (or action, or event) to specify there must be typed. To specify the property, double-click <EMPTY>, type the property name, and press *Enter*. Appendix B lists available properties.

The following example shows how to create a scroll bar that changes the active notebook's zoom factor, using a SEND link command:

1. Create a new dialog window, then create a vertical scroll bar control. Drag a bottom handle of the scroll bar to make it longer.

2. The Zoom Factor property accepts values between 25 and 200; to restrict the scroll bar's value to integers between 25 and 200, right-click the scroll bar, choose Parameters, set Min to 25 and set Max to 200. Choose OK to save the restrictions.
3. Choose Dialog | Links, then Add. To run a link command whenever the scroll bar's value changes, point to the event pick list button (Init appears on it), hold down the mouse button, drag the pointer to Valuechanged, then release.
4. To send the value of the scroll bar to the notebook property Zoom Factor, point to the action pick list button (RECEIVE appears on it), hold down the mouse button, and drag the pointer to SEND. Release the mouse button.
5. When SEND is the link command, the pick list button to the right of the action pick list button specifies which property setting to send. Point to the property pick list button (Value appears on it) and hold down the left mouse button. Notice that all the scroll bar properties are listed in the pick list. Since it's already set to Value in this case (the property setting you want to send), just release the mouse button.
6. To change a property of the active notebook, point to the object pick list button (<POINT> appears on its face), hold down the left mouse button, drag the pointer to Active_Notebook, then release the mouse button.
7. The pick list button to the right of an object pick list button specifies the property to change. Point to it and hold down the left mouse button. Notice all the properties of a notebook are listed. Drag the pointer to Zoom_Factor and release the mouse button.
8. The link command is ready and should look like the next figure:

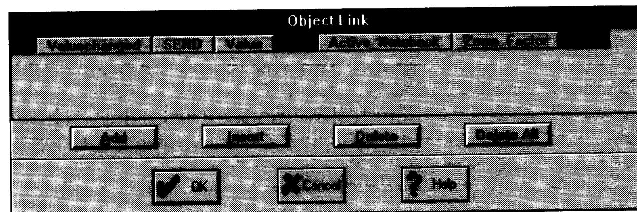


Figure 6.31
A link command that will change the zoom factor of the notebook

9. Choose OK.

10. Choose the Test button from the SpeedBar to test the dialog box's operation.
11. Drag the scroll box down. Notice that the zoom factor of the notebook increases as the scroll box is moved. Choose OK to exit Test mode and return to the dialog window.

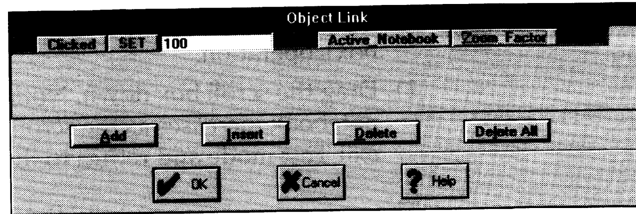
The next example expands on the previous example by adding a push button that resets the zoom factor back to 100 using a SET link command.

Example 3:
A link command that resets the zoom factor to 100%

1. In the dialog window created in the previous example, choose the Push Button tool and click somewhere in the dialog box. Double-click the button, type `Reset`, and press *Enter*.
2. Choose Dialog | Links, then choose Add to create a link command.
3. To make the link command run whenever the user clicks the button, point to the event pick list button (Init appears on it), hold down the left mouse button, drag to Clicked, then release.
4. The SET link command sets a specific property to a specific setting (instead of copying a setting, like SEND does). Point to the action pick list button (RECEIVE appears on it), hold down the left mouse button, and drag to SET. Release the mouse button. An edit field appears next to the pick list button.
5. Enter 100 into the edit field. This specifies that the setting to send to the other object is 100.
6. Point to the object pick list button (<POINT> appears on it), hold down the left mouse button, drag to Active_Notebook and release. This specifies that the property this command should set is in the active notebook.
7. Point to the last pick list button, hold down the left mouse button, drag to Zoom_Factor and release. This specifies that the Zoom Factor should be set to 100 when the user clicks this button.

   The link command should look like the next figure.

Figure 6.32
A link command that will set the notebook's Zoom factor to 100%



8. Choose OK.

To test the control,

1. Choose the Test button from the SpeedBar.
2. Drag the scroll box down to increase the zoom factor of the notebook.
3. Click the push button labeled Reset. Notice the notebook snaps back to the default zoom factor (100%).
4. Choose OK to exit Test mode.

Example 4:
A link command that reads the alignment of a specific cell

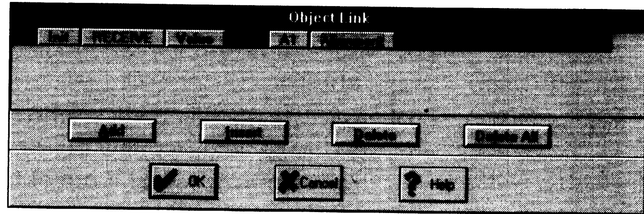
The RECEIVE link command is structured like SEND, but it copies a property setting *from* the other object to a property in the object containing the link command. RECEIVE link commands are useful for getting information when the dialog box initially appears. This example shows how to create an edit field that gets the alignment of a specific notebook cell.

1. Choose Tools | UI Builder to create a new dialog window.

2. Choose the Edit Field tool and click in the dialog window to create an edit field. Drag a right handle of the edit field to make it wider.
3. Choose Dialog | Links, then choose Add to create a link command.
4. To change a property of a specific cell, point to the object pick list button (<POINT> appears on it), hold down the left mouse button for a moment and release it to enter POINT mode. Click on cell A1 of the active notebook page; A1 displays on the object pick list button.
5. The property setting to receive is stored in the Alignment property, so point to the pick list button to the right of the object pick list button and hold down the left mouse button. Drag the mouse pointer to Alignment and release the mouse

button. The link command should look similar to the next figure.

Figure 6.33
A link command that will read a cell's alignment



6. Choose OK.

7. Choose the Test button from the SpeedBar to test the dialog box's operation.
8. Notice the edit field now contains General (the current setting of cell A1's Alignment property). Choose OK to exit Test mode and return to the dialog window.

Example 5:
A link command that displays another dialog box

Link commands can make objects perform *actions*. For example, a link command can activate a control or make a dialog box appear.

The EXECUTE link command makes dialog controls and dialog boxes perform actions. The following table lists the actions that link commands can perform on another dialog box or dialog control.

Table 6.21
Dialog box/control actions

Action	Definition
Clone	Duplicates the object.
Delete	Removes the object (equivalent to Edit | Clear).
Execute	If the object is a dialog box, Execute displays it. If the object is a button, Execute makes the button act as though it was clicked. Other dialog controls are unaffected.
Move_top	Moves the object in front of any objects covering it.
Move_bot	Moves the object behind any objects it's covering.
Activate	Activates a control for manipulation.
Deactivate	Deactivates the control.

This example shows how to use EXECUTE to display another dialog box:

1. Choose Tools | UI Builder to display a new dialog box.
2. Right-click the dialog box and set Name to `Warning`.
3. Create a label. Double-click the label and enter `Lose Your Changes?`. Drag the label to just above the OK and Cancel buttons.
4. Activate the notebook window and choose Tools | UI Builder to display another new dialog window. Minimize the notebook window to put it out of the way. Move the active dialog window until both dialog windows appear.
5. Select the OK Button (in the new dialog window, not the Warning dialog window) and choose Dialog | Links.
6. Choose Add to create the link command. Point to the event pick list button (Init appears on it) and hold down the left mouse button. Drag the pointer to Clicked and release the mouse button. This specifies that the link command runs when the user clicks the button.
7. Point to the action pick list button (RECEIVE appears on it) and hold down the left mouse button. Drag the pointer to EXECUTE and release the mouse button to save the setting.
8. Point to the object pick list button (<POINT> appears on it) and hold down the left mouse button. Drag the pointer to <POINT> and release the mouse button. Click in the other dialog window (the one previously named Warning). `Warning:` appears on the pick list button.
9. In an EXECUTE command, the action to perform is specified using the pick list button to the right of the object pick list button. Choose execute from this pick list (clone appears on it) to specify that the Warning dialog box should appear when the user clicks the OK button in this dialog box. The link command should now look like the next figure:

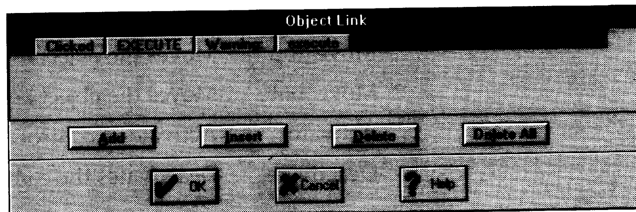


Figure 6.34
A link command that will display another dialog box

10. Choose OK to save the link command.

11. Choose the Test button from the dialog SpeedBar to test the dialog box's operation.
12. Click the OK button; normally this closes the dialog box, but in this case the Warning dialog box appears first. Choose OK to close the Warning dialog box. This also closes the first dialog box, exits Test mode, and returns you to the dialog window.

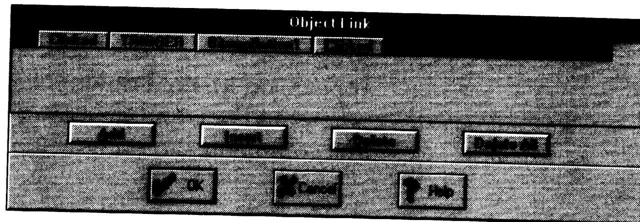
Example 6: A link command that simulates events

The TRIGGER command fools another object into thinking that a certain event occurred; the object then runs any link commands that normally respond to the event. Here's an example:

1. Choose Tools | UI Builder to create a new dialog window.
2. Click the OK Button and choose Dialog | Links.
3. Choose Add to create a link command. Set the event pick list button (Init appears on it) to Clicked, the action pick list button (RECEIVE appears on it) to DOMACRO, and enter {BEEP}{BEEP} in the edit field provided. Choose OK to save the link command.

4. Choose the Push Button tool and click in the dialog window to create a push button.
5. Double-click the Push Button, type `Beeper`, and press *Enter* to rename the button.
6. Choose Dialog | Links, then choose Add to create a link command for the push button. Set the event pick list button to Clicked and the action pick list button to TRIGGER.
7. Choose <POINT> from the object pick list and click the OK button in the dialog window (you may need to move the Links dialog box out of the way to see the OK button in the dialog window).
8. The pick list button to the right of the object pick list button now contains a pick list of events to emulate. Set this pick list button to Clicked. The link command should now look like the next figure:

Figure 6.35
A link command that will simulate an event



9. Choose OK to save the link command.
10. Choose the Test button from the Dialog SpeedBar to test the dialog box's operation.
11. Click the push button named Beeper. Two beeps sound because the push button's link command makes the link command in the OK button run. Notice the dialog box doesn't close, because TRIGGER runs only link commands.
12. Click the OK Button. Two beeps sound, but this time Quattro Pro exits Test mode and returns you to the dialog window.

See page 351 for another example of using a TRIGGER link command.

A time control example

This example shows how to use a time control to create a SpeedBar that tracks the name of the active block's typeface:

1. Select the Graphs page.
2. Click the New SpeedBar tool on the Graphs page SpeedBar to display a dialog window. This is the SpeedBar in an editable form.

3. Choose the Edit Field tool from the dialog SpeedBar and click in the SpeedBar to create an edit field. Drag a right handle of the edit field to widen it.
4. Choose Dialog | Links, then choose Add to add a link command to the edit field.
5. Set the event pick list button (Init appears on it) to Trigger. This allows the control to get the active block's typeface, and prevents the user from being able to run the link command that does this.
6. To receive a setting from the active block, set the object pick list button (<POINT> appears on it) to Active_Block. The setting to receive is stored in the active block's Font property, so point to the property pick list button (Numeric_Format appears on it), hold down the left mouse button, drag the pointer to Font, and then drag the pointer to TypeFace. The link command should now look like the next figure.

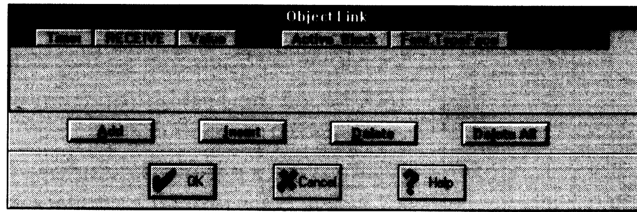


Figure 6.36
A link command that will track the typeface of the active block

7. Choose OK to save the link command and return to the dialog window.
8. Choose Dialog | Save SpeedBar As, type `TIMETRAK.BAR`, and then choose OK to save your work.

9. Choose the Time Control tool from the dialog SpeedBar and click in the dialog window to create a time control.
10. To generate Timer events as quickly as possible, right-click the time control and set Interval In Units to 1. Right-click the time control again and set Units In Milliseconds to 10. This makes the time control generate a Timer event every hundredth of a second. Right-click the time control a final time and set Timer On to Yes to enable the timer.
11. To run the edit field's link command at regular intervals, you can use TRIGGER. Choose Dialog | Links, then choose Add to create a link command.
12. Set the event pick list button (Init appears on it) to Timer to run this link command whenever the previously specified interval has elapsed. Set the action pick list button (RECEIVE appears on it) to TRIGGER to trigger another link command when the event occurs. Choose <POINT> from the object pick list and click the edit field on the SpeedBar. Finally, set the pick list button to the right of the object pick list button to `Trigger`. The link command should now look like the next figure.

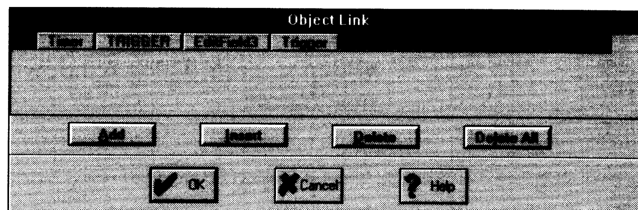
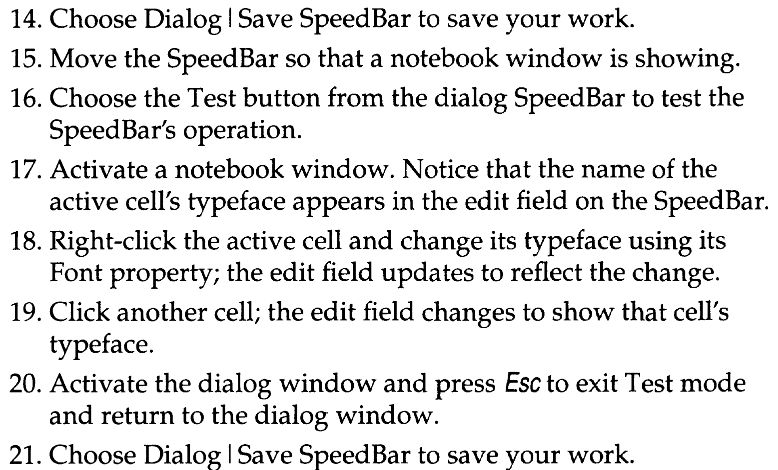


Figure 6.37
A link command that will update an edit field at specific time intervals

13. Choose OK.



14. Choose Dialog | Save SpeedBar to save your work.
15. Move the SpeedBar so that a notebook window is showing.
16. Choose the Test button from the dialog SpeedBar to test the SpeedBar's operation.
17. Activate a notebook window. Notice that the name of the active cell's typeface appears in the edit field on the SpeedBar.
18. Right-click the active cell and change its typeface using its Font property; the edit field updates to reflect the change.
19. Click another cell; the edit field changes to show that cell's typeface.
20. Activate the dialog window and press *Esc* to exit Test mode and return to the dialog window.
21. Choose Dialog | Save SpeedBar to save your work.

Dialog box examples

This section lists small examples of adding controls and link commands to dialog boxes.

Example 1: Connecting a file control to an edit field

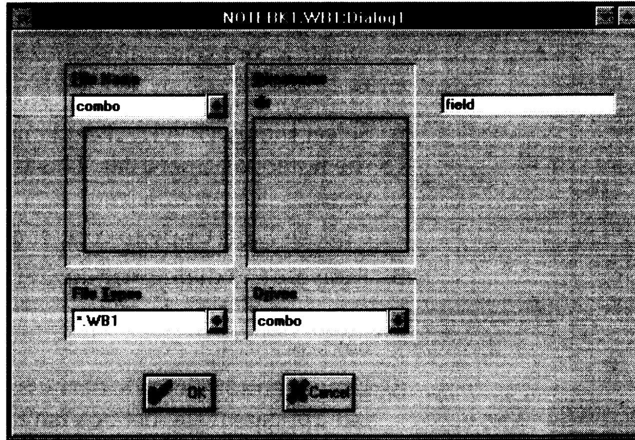
This example shows how to create a file control that gets its value from an edit field:

1. Choose Tools | UI Builder to display a new dialog window. Drag the lower right corner of the window to enlarge it.

2. Choose the File Control tool from the dialog SpeedBar and click in the upper left corner of the window to create the file control.

3. To create an edit field, choose the Edit Field tool from the SpeedBar and click in the dialog window to the right of the file control. Drag a handle on the right side of the edit field to widen it. Your dialog window should now look similar to the next figure:

Figure 6.38
The dialog window after adding a file control and an edit field



4. Choose Dialog | Connect. Choose Target, then click the file control. Choose OK to create the link between the controls. Now whenever the dialog box displays, the edit field is set to the value of the file control. While the dialog box appears, changing the edit field's setting also sends the edit field's new value to the file control. (See page 339 for more information on Dialog | Connect.)

5. Choose the Test button from the SpeedBar to test the dialog box's operation. Notice the edit field is set to the path and wildcard displayed in the file control.

If you don't keep your DOS files in the directory DOS on drive C: of your system, enter the path to your DOS files instead of C:\DOS\ for this step.

6. Click the edit field and enter `C:\DOS*.EXE`. Notice that the file control changes to display your DOS directory and the .EXE files located in it.
7. Delete *.EXE from the edit field and enter `CHKDSK.*` after the backslash. Notice that CHKDSK.EXE appears in the file control (below File name).
8. Choose OK to exit Test mode and return to the dialog window.

Example 2: Creating a list box

This example shows how to create a list box that shows the days of the week:

1. Choose Tools | UI Builder to display a new dialog window.

2. Choose the List Box tool from the dialog SpeedBar.
3. Click in the dialog window to create the list box.

4. To put the days of the week into the list, activate a notebook window and select an empty notebook page.
5. Enter `Monday` in cell A1.
6. Select the block A1..A7 and choose the SpeedFill button from the notebook SpeedBar. This fills the block with the days of the week.
7. Choose Edit | Copy to copy the block to the Clipboard.
8. Activate the dialog window and select the list box.
9. Choose Edit | Paste. The days of the week now appear in the list box.
10. Drag a bottom handle of the list box up. Notice that a scroll bar appears if the list box isn't large enough to display all the items.

Example 3: Creating a history list (combo box)

Here's an example of configuring a combo box as a history list:

1. Choose Tools | UI Builder to display a new dialog window.

2. Choose the Combo Box tool and then click the dialog window to create the combo box.
3. To make the combo box a history list, right-click the combo box and set the History List property to Yes.
4. To specify the number of entries to retain, right-click the combo box again and set the List Length property to 4. Whenever the list exceeds this length, the oldest entries are removed. The combo box in this example retains the last four entries.
5. To prevent the dialog box from closing when a user presses *Enter* in the dialog box's edit field, right-click the combo box and set Terminate Dialog to No.

6. Choose the Test button from the SpeedBar to test the dialog box's operation.
7. Click the combo box's down arrow to display its list. The list box is empty.
8. Click the combo box's edit field, type `Francs` and press *Enter*. Click the down arrow button to see that Francs now appears in the list.

9. Click the combo box's edit field again. Delete Francs and enter `Marks`. Click the down arrow button. Marks and Francs appear in the list.
10. Click the combo box, then delete Marks and enter `Yen`. Click the down arrow button. Yen, Marks, and Francs appear in the list.
11. Click the combo box, then delete Yen and enter `Pesos`. Click the down arrow button. Pesos, Yen, Marks, and Francs appear in the list.
12. Click the combo box's edit field., then delete Pesos and enter `Dollars`. Click the down arrow button. Dollars, Pesos, Yen, and Marks appear in the list box. Francs was removed, since the List Length property is set to 4.
13. Choose Marks from the list; Marks now appears in the edit field.
14. Choose OK to exit Test mode and return to the dialog window.

C H A P T E R

7

Menus

Quattro Pro offers a variety of macro commands to add and delete menus and menu commands from the active menu bar. This chapter provides an overview of Quattro Pro menus, and explains how to

- create menus and menu commands
- add and delete menus from the menu bar
- replace the menu bar using the {SETMENUBAR} command.
- define commands in a menu using command definitions
- add and delete menu commands from the menu bar
- design a menu

You can also use the commands {MENUBRANCH} and {MENUCALL} to produce popup menus that disappear once an item is selected. See Chapter 4 for a description of these commands.

What's a menu?

A *menu* is a collection of onscreen lists that manipulate an application. Each list contains *menu commands* the user can choose to perform operations (such as saving a file) or display other menus (called *submenus* or *cascading menus*). The following table lists terms associated with menus.

Table 7.1: Menu terms

Term	Meaning
menu bar	A line at the top of the application window that displays menu commands.
menu	An onscreen list of items the user manipulates to communicate with the program.
menu command	An item on a menu the user can choose to perform actions (such as displaying a dialog box, saving a file, or displaying another menu).
menu path	The sequence of menu commands chosen to perform an action or display a dialog box. To enter menu paths in a macro command, enter the sequence of menu commands separated by forward slashes. For example, /File/Save As.
dimmed	Describes a menu command that shows in a lighter color (usually gray instead of black) and isn't selectable. Also known as an *unavailable* or *grayed* menu command.
divider line	A horizontal line in a menu that separates groups of menu commands.
hint	A line of text appearing on the status line to provide help when a menu command is highlighted.
link command	An action performed by a menu command when chosen. (See page 352 for a complete discussion of link commands.)
shortcut key	A keystroke listed to the right of a menu command's name that the user can press to choose it. Also known as an *accelerator*.
underlined letter	An underlined character in a menu command's name. The user can press this letter to choose the menu command (if its menu is displayed). Also known as a *mnemonic*.

Included with Quattro Pro is a file called QUIKSAVE.WB1. This file contains a macro that replaces the standard File | Save command. The next figure shows the revised File | Save menu (as added by a macro):

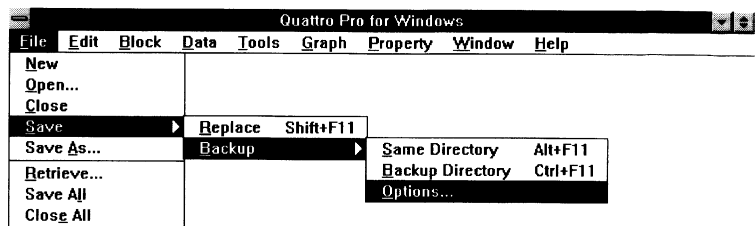
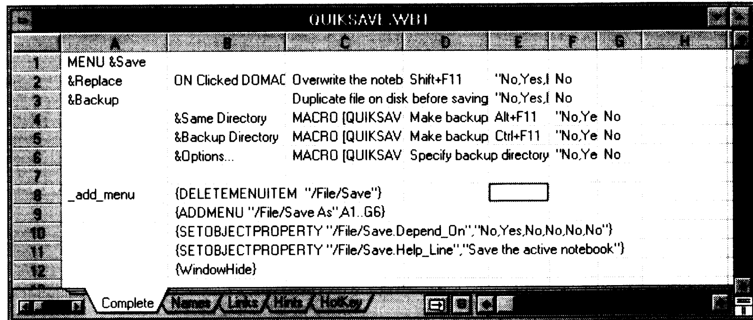


Figure 7.1
A new File | Save menu command

The notebook page named Complete in QUIKSAVE.WB1 contains the macro and information that creates the new menu, as shown in the next figure.

Figure 7.2
A menu block and macro



The remainder of this chapter uses QUIKSAVE.WB1 to help explain creating menus and menu commands.

Adding menus to the menu bar

Changing the menu bar is a process of removing unneeded menus and menu commands with {DELETEMENU} and {DELETEMENUITEM}, then adding your own with {ADDMENU} and {ADDMENUITEM}. The macro _add_menu in Figure 7.2 uses {DELETEMENUITEM "/File/Save"} to remove the standard Save command from the File menu. The command {ADDMENU "/File/Save As",A1..G6}} adds the new Save command to the File menu ("/File/Save As" indicates that the menu is to be inserted above Save As on the File menu).

{ADDMENU} uses a *menu block* to construct the menu added. The menu block contains labels that define the menu; the block A1..G6 in the previous figure is a menu block. The first row of the menu block contains the title of the menu to create. This title is preceded by MENU. In the previous example, MENU &Save specifies that the name of this menu is Save. Each row in the menu block (except for the first) defines one menu command using a *command definition*. Each cell within a command definition specifies one aspect of the command to add; every command definition is a row of six adjacent cells. The contents of these cells are discussed on page 374.

The first row of a menu block defines the menu command that displays the menu. Any command definitions that begin in the first column of the menu block display on this menu. In Figure

7.2, A1 defines the command that displays the menu as Save. The commands Replace and Backup (defined by the labels in row A2..F2 and A3..F3) are on the Save menu because their definitions start in the same column as MENU &Save.

To make a command in the menu block display a menu, start the command definitions below it one column to its right. For example, in Figure 7.2, starting the definitions of Same Directory, Backup Directory, and Options in the second column of the menu block has these results:

- makes Backup a command that displays a menu
- places Same Directory, Backup Directory, and Options on the Backup menu

Starting the command definition of Backup Directory in the third column would make Same Directory a menu containing the Backup Directory command. See Chapter 4 for more details on {ADDMENU} and other UI-building commands.

Replacing the menu bar

You can make a menu block the active menu bar using {SETMENUBAR}, which has one argument (the menu block). The commands on the menu replace any commands on the menu bar; the command defined in the first row of the menu block is ignored. To restore the original menu bar, use the command {SETMENUBAR}. If Quattro Pro is in Developer mode (see Chapter 5) you can press *Ctrl+Shift+N* to restore the original menu bar.

Command definitions

This section describes each of the six settings (cells) that make a command definition. You can study QUIKSAVE.WB1 to see examples of these settings. Because many of these cells contain long labels, QUIKSAVE.WB1 breaks down the menu block setting-by-setting, starting with the notebook page Names. The page Complete shows the actual menu block used by the macro _add_menu.

Naming a menu command

The first cell of a command definition contains the name of the command. Its underlined letter is preceded by an ampersand (&). (The user can press this key to choose the command.) For example, entering `&File` makes the menu command name display as `File` in the menu. The next figure shows the cells that define menu command names for the menu block of Figure 7.2:

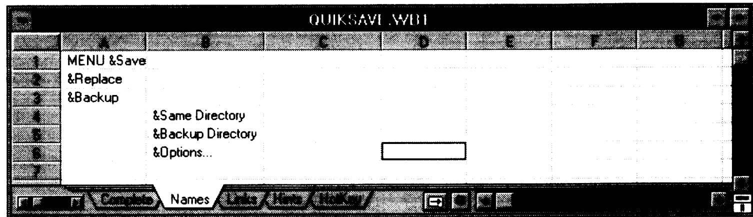


Figure 7.3
Names in a menu block

You can use {SETOBJECTPROPERTY} to change a menu command's name after it's created. See page 380.

Making menu commands act (Link commands)

The second cell of a command definition contains a link command similar to those in dialog boxes (see Chapter 6) or the name of a macro to run when the menu command is chosen.

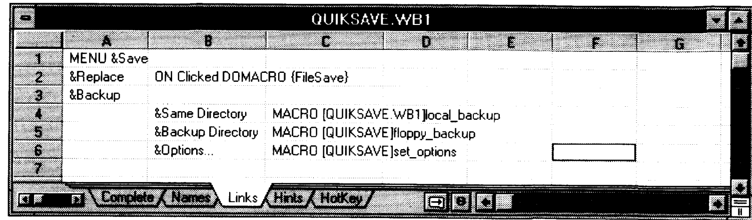
To run a macro stored in a notebook, enter `MACRO` *MacroName* into the cell, where *MacroName* is the name of the macro to run. For example, in Figure 7.2 the menu command Backup Directory uses the label `MACRO [QUIKSAVE]floppy_backup` to make the menu command run the macro floppy_backup (in QUIKSAVE.WB1) when chosen.

Menu commands can display the dialog boxes in Quattro Pro's normal menus using command equivalents (see page 129). For example, the following link command makes the menu command behave like Block | Copy:

```
ON Clicked DOMACRO {BlockCopy?}
```

The next figure shows the link commands used in QUIKSAVE.WB1:

Figure 7.4
Link commands in a menu block



In menu commands, a link command is typed instead of specified with a dialog box (see 352 for information on link commands in dialog boxes, which are created with Dialog | Links). The following table lists the events a menu command's link command can detect.

Table 7.2
Events a menu command can detect

Event	**Description**
Clicked	The user chose the command from the menu.
Init	The menu containing the menu command is about to display.
Activate	The menu command has just been highlighted.
Deactivate	The menu command has just been unhighlighted.

You can attach multiple link commands to the same menu command; separate them with commas in the cell. If you don't want to assign any actions to a menu command, leave the second cell of its definition blank.

The remainder of this section lists the syntax of link commands that you can enter into a menu block. In each link command, *event* is one of the events listed in Table 7.2, *Object.ObjectProperty* is a string identifying the object and property setting to change or read, and *Property* is the name of a property in the menu command. See Appendix B for more information on the syntax of *Object.ObjectProperty* and *Property*.

EXECUTE

Format: ON *event* EXECUTE *Object.Action*

Performs *Action* on *Object* when *event* occurs. *Object* is a dialog box or dialog control. See Chapter 6 for more information on *Action* and the EXECUTE link command.

DOMACRO

Format: ON *event* DOMACRO *MacroText*

Runs the macro *MacroText* when *event* occurs. *MacroText* is a series of macro commands. The following link command saves the active notebook when the menu command is chosen:

```
ON Clicked DOMACRO {FileSave}
```

RECEIVE **Format:** ON *event* RECEIVE *Property* FROM *Object.ObjectProperty*

Sets the menu command's *Property* property to *Object's* *ObjectProperty* when *event* occurs. *ObjectProperty* and *Property* don't have to be the same. The following link command enables the menu command whenever Tools | Macro is enabled:

```
ON Init RECEIVE Enabled FROM /Tools/Macro.Enabled
```

SEND **Format:** ON *event* SEND *Property* TO *Object.ObjectProperty*

Sends the current setting of the menu command's *Property* property to *Object's* *ObjectProperty* when *event* occurs. *ObjectProperty* and *Property* don't have to be the same. The following link command stores the menu command's title in the current cell:

```
ON Clicked SEND Title TO Active_Block.Value
```

SET **Format:** ON *event* SET *Setting* TO *Object.ObjectProperty*

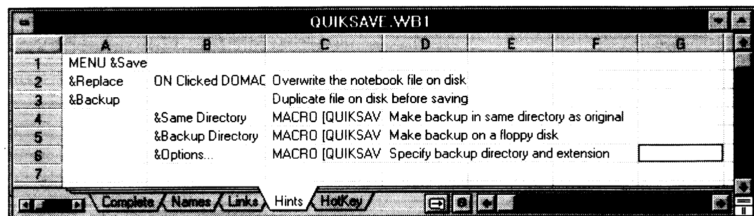
Sets *Object's* *ObjectProperty* to *Setting* when *event* occurs. *Setting* is a value appropriate to the property (for example, Yes or Beveled Out). The following link command sets the active notebook's zoom factor to 150 percent:

```
ON Clicked SET 150 TO Active_Notebook.Zoom_Factor
```

Menu hints

The third cell of a command definition contains a brief help message that displays on the status line when the menu command is highlighted. Use these hints to make the menus easier to understand. The next figure shows the hints in QUIKSAVE.WB1:

Figure 7.5 Adding hint text to menu commands


QUIKSAVE.WB1

	A	B	C	D	E	F	G
1	MENU &Save						
2	&Replace	ON Clicked DOMAC	Overwrite the notebook file on disk				
3	&Backup		Duplicate file on disk before saving				
4		&Same Directory	MACRO [QUIKSAV	Make backup in same directory as original			
5		&Backup Directory	MACRO [QUIKSAV	Make backup on a floppy disk			
6		&Options...	MACRO [QUIKSAV	Specify backup directory and extension			
7							

Complete / Names / Links / Hints / HotKey

Assigning a shortcut key

The fourth cell of a command definition contains a shortcut key that the user can press to choose the command. If an item doesn't need a shortcut key, leave the fourth cell blank.

Separate each component of the keystroke with the plus sign (+). For example, if a shortcut key is comprised of *Alt* and the *F2* key, enter it as `Alt+F2`. If a shortcut key is used by Quattro Pro for some other operation, that operation occurs first. See the keyboard template included with Quattro Pro to find unassigned function keys. The next figure shows the shortcut keys in QUIKSAVE.WB1:

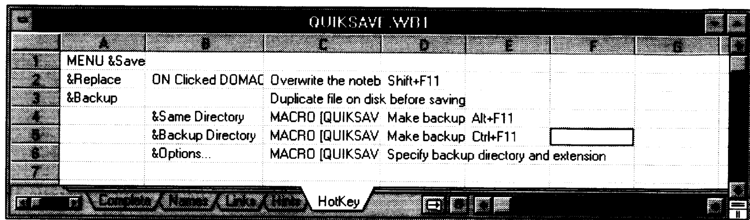


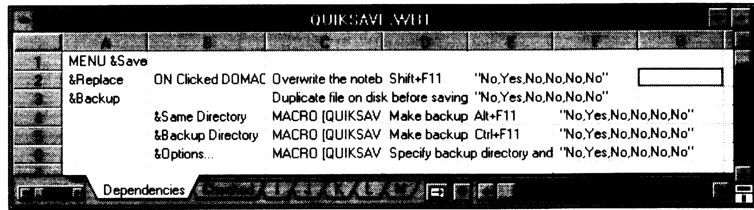
Figure 7.6
Adding shortcut keys to a menu block

You can assign shortcut keys to existing commands using {SETOBJECTPROPERTY}; see page 380.

Dimming menu commands

The fifth cell of a command definition specifies the areas of Quattro Pro in which the menu command is available. Enter `Yes` or `No` for each area, separated by commas, in the following order: *Desktop, Notebook, Graph window, Dialog window, Input line, Graphs Page*. The string must be enclosed in quotes. A menu command is dimmed when it isn't available and the user can't select it. To make a command available in all areas, leave this cell of the definition blank. The new Save menu of Figure 7.2 is available only when a notebook window is active, as shown in the next figure:

Figure 7.7
Specifying when the command is available



You can also disable an existing menu command with {SETOBJECTPROPERTY}; see page 380.

Placing check marks by menu commands

The sixth cell of a command definition contains `Yes` if the menu command should appear checked when it's displayed. If this cell is blank (or contains `No`), the item displays unchecked. You can check or uncheck a menu command after its creation using {SETOBJECTPROPERTY}; see page 380.

Deleting menus

You can remove a menu from the menu bar using {DELETEMENU}, or hide it using {SETOBJECTPROPERTY}. For example, the following macro removes the Data menu:

```
{DELETEMENU "/Data"}
```

See Chapter 4 for more information on {DELETEMENU}. See page 380 for information on using {SETOBJECTPROPERTY} to hide menu commands.

Adding and deleting menu commands

{ADDMENUITEM} uses the same information as a command definition but adds a single menu command to the menu bar. Each of the six settings discussed earlier is passed as a separate argument to {ADDMENUITEM}. See Chapter 4 for a complete description of {ADDMENUITEM}.

You can use {DELETEMENUITEM} to remove a single menu command from the menu bar. {DELETEMENUITEM} removes only menu commands that don't display a menu; use {DELETEMENU} to

remove menus. You can hide a menu command using {SETOBJECTPROPERTY}.

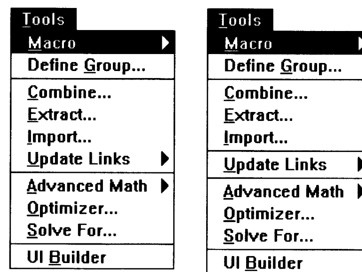
Divider lines

You can use divider lines to group related commands in a menu. The horizontal line between Define Group and Combine in the Tools menu is an example of a divider line. To insert a divider line in a menu block, insert a row between the definitions of the two menu commands it divides. In this row (and in the same column as the names of the command being divided) enter eight hyphens (-) as a label. You can also add divider lines with {ADDMENUITEM}. For example, to place a divider line before Update Links in the Tools menu, use

{ADDMENUITEM "/Tools/Update Links","--------"}

The next figure shows the effect of this command. The left menu is the Tools menu before adding the divider line; the right menu is the Tools menu after the divider line is added.

Figure 7.8
Adding divider lines



Changing command properties

Each command on the menu bar has properties you can manipulate. To identify a menu command in @PROPERTY or a macro command, enter the menu path separated by forward slashes (/). (See Appendix B for the full syntax of a menu path.) Don't include ellipses (...). For example, the following formula returns whether Edit | Paste Format is dimmed on the menu:

@PROPERTY("/Edit/Paste Format.Grayed")

Each menu command has the following properties:

- **Title** is set to the name of the menu command; the command's underlined letter is preceded by an ampersand (&). Use this property to rename a command or change its underlined letter. For example, the following command changes the underlined letter of File to i:

  {SETOBJECTPROPERTY "/File.Title","F&ile"}

- **Checked** is set to Yes when a check mark is displaying by the command. Use this property to check and uncheck menu commands. For example, the following command checks Edit | Copy:

  {SETOBJECTPROPERTY "/Edit/Copy.Checked","Yes"}

- **HotKey** is set to the shortcut key that you want to invoke the command with. For example, the following macro assigns *Shift+F11* to File | Save:

  {SETOBJECTPROPERTY "/File/Save.HotKey","Shift+F11"}

- **Help line** is set to the hint that appears on the status line. For example, the following command removes the hint from Graph:

  {SETOBJECTPROPERTY "/File/Save.Help_Line",""}

- **Depend On** is set to the areas in which the menu command is available, using the same syntax as dependencies in a menu block (see page 378). For example, the following command makes File | Close available everywhere except the desktop:

  {SETOBJECTPROPERTY "/File/Close.Depend_On",
  "No,Yes,Yes,Yes,Yes,Yes"}

- **Grayed** is set to Yes to dim the menu command and disable it. For example, the following command dims Data:

  {SETOBJECTPROPERTY "/Data.Grayed","Yes"}

- **Enabled** is set to No to disable a menu command without dimming it. For example, the following command enables Data:

  {SETOBJECTPROPERTY "/Data.Enabled","Yes"}

- **Disabled** behaves like Enabled, but it's set to No to enable the menu command and Yes to disable it.

- **Hidden** is set to Yes to hide the menu command. For example, the following command hides Tools | Advanced Math:

  {SETOBJECTPROPERTY
  "/Tools/Advanced Math.Hidden","Yes"}

- **Show** is the same as Hidden, but it's set to No to hide the menu command and Yes to display it.

See Chapter 4 and Appendix B for more information on using @functions and macro commands to manipulate menu command properties.

Designing a menu

Building a menu is fairly easy, but designing one can present challenges to the novice UI builder.

Here are some design guidelines for creating a menu:

- **Keep menus short.** Users lose track after about eight menu commands. Use submenus to put more commands in a menu.
- **Make menus on the menu bar shallow**. Count how many steps it takes to choose a command; keep it down to two or three.
- **Group menu commands logically.** Keep related commands near each other (for example, don't put Save All in the Windows menu when Save is in the File menu). If menu commands are often chosen in sequence, arrange them in the order you choose them, from the top down. Use divider lines to separate groups visually.
- **Put the most common menu commands at the top.** Putting common menu commands at the top makes them easier to find.
- **Keep it short.** The text of menu commands should be as short as possible without being cryptic.
- **Try to make the first letters of menu commands unique.** Then you can make the first letter of each menu command its underlined letter (see page 372). This is easier for the user to remember.
- **Use ellipses (...) correctly.** If a menu command leads to a dialog box, its text should end with an ellipsis. By Windows convention, an ellipsis means a dialog box follows, both in menu commands and in dialog boxes.
- **Dim menu commands only when appropriate.** A menu command should be dimmed when it's irrelevant to what the user is currently doing. See the description of {SETOBJECTPROPERTY} in Chapter 4 for information on dimming menu commands.
- **Don't tamper with File, Edit, and Help too much.** Windows users expect a consistent set of items in these menus.

A P P E N D I X

A

Command equivalents

Quattro Pro records a variety of operations as special macro commands called *command equivalents*. The following table lists each menu command and its command equivalents. Items in bold are menu names; underlined items are dialog box controls. Items in italics are arguments that are specified when entering the command equivalent; the italicized words describe what information is expected. If an argument ends with a question mark (?), a yes or no answer is expected; enter 1 for yes or 0 for no. Arguments enclosed in angle brackets (<>) are optional. You can use many of the command equivalent names with @COMMAND to find current Quattro Pro settings. For example, the following formula returns the current setting of Find in the Edit | Search and Replace dialog box:

@COMMAND("Search.Find")

See Chapter 2 for a description of @COMMAND. See page 129 and Chapter 4 for more information on using command equivalents.

Table A.1: Command equivalents by command

Command	Command equivalent
Control Menu	
Restore	{WindowRestore}
Move	{WindowMove *UpperLeftX*, *UpperLeftY*}
Size	{WindowSize *LowerRightX*, *LowerRightY*}
Minimize	{WindowMinimize}
Maximize	{WindowMaximize}

Table A.1: Command equivalents by command (continued)

Command	Command equivalent
Close	{WindowClose} {FileExit <*doSave?*>} closes the Quattro Pro window
Next	{WindowNext}
File	
New	{FileNew}
Open	{FileOpen *Filename*}
Close	{FileClose <*doSave?*>}
Save	{FileSave <"Replace" or "Backup" or "Confirm">}
Save As	{FileSaveAs *Filename* <,"Replace" or "Backup" or "Confirm">}
Retrieve	{FileRetrieve *Filename*}
Save All	{FileSaveAll <"Replace" or "Backup" or "Confirm">}
Close All	{FileCloseAll <*doSave?*>}
Print Preview	{Preview}
Page Setup	
Header	{Print.Header *HeaderString*}
Footer	{Print.Footer *FooterString*}
Margins	
Top	{Print.Top_Margin *Value*}
Bottom	{Print.Bottom_Margin *Value*}
Left	{Print.Left_Margin *Value*}
Right	{Print.Right_Margin *Value*}
Header	{Print.Header_Margin *Value*}
Footer	{Print.Footer_Margin *Value*}
Header Font	{Print.Headers_Font "*Typeface, PointSize, Bold?, Italic?, Underline?, Strikeout?*"}
Options	
Break pages	{Print.Page_Breaks "Yes" or "No"}
Print to fit	{Print.Print_To_Fit "Yes" or "No"}
Center blocks	{Print.Center_Block "Yes" or "No"}
Paper type	{Print.Paper_Type *PaperSize*}
Scaling	{Print.Scaling *PercentageValue*}
Print orientation	{Print.Orientation "Landscape"} {Print.Orientation "Portrait"}
Reset Defaults	{Print.PageSetupReset} {Print.PrintReset} resets all print settings
Print	
Print blocks	{Print.Print_Block *Blocks*}
Print pages	
All pages	{Print.All_Pages "Yes" or "No"}
From	{Print.Start_Page_Number *Value*}
To	{Print.End_Page_Number *Value*}
Copies	{Print.Copies *Value*}
Preview	{Preview}
Options	
Top heading	{Print.Top_Heading *Block*}
Left heading	{Print.Bottom_Heading *Block*}

Table A.1: Command equivalents by command (continued)

Command	Command equivalent
Print options	
Cell formulas	{Print.Cell_Formulas "Yes" or "No"}
Gridlines	{Print.Print_Gridlines "Yes" or "No"}
Row/Column borders	{Print.Print_Borders "Yes" or "No"}
Print between blocks	
Lines	{Print.Between_Block_Formatting "Lines"}
	{Print.Lines_Between_Blocks *Value*}
Page advance	{Print.Between_Block_Formatting "Page Advance"}
Print between 3D pages	
Lines	{Print.Between_Page_Formatting "Lines"}
	{Print.Lines_Between_Pages *Value*}
Page advance	{Print.Between_Page_Formatting "Page Advance"}
Reset Defaults	{Print.PrintOptionsReset}
	{Print.PrintReset} resets all print settings
Print	{Print.DoPrint}
	{Print.DoPrintGraph} prints the graph showing in the selected floating graph, graph icon, or graph window
Printer Setup	{PrinterSetup *Printer, Port, PrintToFile?, Filename, ReplaceOptions*}
Named Settings	
Named settings	{Print.Use *NamedSetting*}
Create	{Print.Create *NamedSetting*}
Update	{Print.Create *NamedSetting*}
Delete	{Print.Delete *NamedSetting*}
Workspace	
Save	{Workspace.Save *Filename*}
Restore	{Workspace.Restore *Filename*}
Exit	{FileExit *<doSave?>*}
Edit	
Undo	{Undo}
Cut	{EditCut}
Copy	{EditCopy}
Paste	{EditPaste}
Clear	{EditClear}
Clear Contents	{ClearContents *<FirstPageOnly?>*}
Paste Link	{PasteLink}
Paste Special	{PasteSpecial "Properties" or "", "Formulas" or "Values" or "", "Transpose" or "", "NoBlanks" or ""}
Paste Format	{PasteFormat *FormatType*}
Goto	{EditGoto *Block*}

Table A.1: Command equivalents by command (continued)

Command	Command equivalent
Search and Replace	
Blocks	{Search.Block *Block*}
Find	{Search.Find *String*}
Replace (edit field)	{Search.ReplaceBy *String*}
Look In	
Formula	{Search.Look_In "Formula"}
Value	{Search.Look_In "Value"}
Condition	{Search.Look_In "Condition"}
Options	
Columns First	{Search.Direction "Column"} to check {Search.Direction "Row"} to uncheck
Match Whole	{Search.Match "Whole"} to check {Search.Match "Part"} to uncheck
Case Sensitive	{Search.Case "Any"} to uncheck {Search.Case "Exact"} to check
Next	{Search.Next}
Previous	{Search.Previous}
Replace (button)	{Search.Replace}
Replace All	{Search.ReplaceAll}
Reset	{Search.Reset}
Define Style	
Define Style For	{NamedStyle.Define *StyleName, Align?, NumericFormat?, Protection?, Lines?, Shading?, Font?, TextColor?*}
Delete	{NamedStyle.Delete *StyleName*}
Merge	None
Included properties	
Alignment	
General	{NamedStyle.Alignment "General"}
Left	{NamedStyle.Alignment "Left"}
Right	{NamedStyle.Alignment "Right"}
Center	{NamedStyle.Alignment "Center"}
Shading	{NamedStyle.Shading "*Color1, Color2, Blend*"}
Format	{NamedStyle.Numeric_Format "*NumericFormat*"}
Font	{NamedStyle.Font "*FontName, PointSize, Bold, Italic, Underline, Strikeout*"}
Protection	{NamedStyle.Protection "Protected" or "Unprotected"}
Text Color	{NamedStyle.Text_Color "*ColorNumber*"}
Line Drawing	{NamedStyle.LineDrawing "*LeftLine, TopLine, RightLine, BottomLine*"}
Insert Object	{InsertObject *ObjectType*}
Block	
Move	{BlockMove *SourceBlock, DestBlock*}
Copy	{BlockCopy *SourceBlock, DestBlock, <ModelCopy?>*}
Insert	
Rows	
Partial	{BlockInsert.Rows *Block*, "Partial"}
Entire	{BlockInsert.Rows *Block*, "Entire"}

Table A.1: Command equivalents by command (continued)

Command	Command equivalent
Columns	
Partial	{BlockInsert.Columns *Block*, "Partial"}
Entire	{BlockInsert.Columns *Block*, "Entire"}
Pages	
Partial	{BlockInsert.Pages *Block*, "Partial"}
Entire	{BlockInsert.Pages *Block*, "Entire"}
File	{BlockInsert.File *FileName, BeforePage*}
Delete	
Rows	
Partial	{BlockDelete.Rows *Block*, "Partial"}
Entire	{BlockDelete.Rows *Block*, "Entire"}
Columns	
Partial	{BlockDelete.Columns *Block*, "Partial"}
Entire	{BlockDelete.Columns *Block*, "Entire"}
Pages	
Partial	{BlockDelete.Pages *Block*, "Partial"}
Entire	{BlockDelete.Pages *Block*, "Entire"}
Fill	
Blocks	{BlockFill.Block *Block*}
Start	{BlockFill.Start *Value*}
Step	{BlockFill.Step *Value*}
Stop	{BlockFill.Stop *Value*}
Order	
Column	{BlockFill.Order "Column"}
Row	{BlockFill.Order "Row"}
Series	
Linear	{BlockFill.Series "Linear"}
Growth	{BlockFill.Series "Growth"}
Power	{BlockFill.Series "Power"}
Year	{BlockFill.Series "Year"}
Month	{BlockFill.Series "Month"}
Week	{BlockFill.Series "Week"}
Weekday	{BlockFill.Series "Weekday"}
Day	{BlockFill.Series "Day"}
Hour	{BlockFill.Series "Hour"}
Minute	{BlockFill.Series "Minute"}
Second	{BlockFill.Series "Second"}
OK	{BlockFill.Go}
Names	
Create	{BlockName.Create *BlockName, Block*}
Delete	{BlockName.Delete *BlockName*}
Labels	
Left	{BlockName.Labels *Block*, "Left"}
Right	{BlockName.Labels *Block*, "Right"}
Up	{BlockName.Labels *Block*, "Up"}
Down	{BlockName.Labels *Block*, "Down"}

Table A.1: Command equivalents by command (continued)

Command	Command equivalent
Reset	{BlockName.Reset}
Make Table	{BlockName.MakeTable *Block*}
Transpose	{BlockTranspose *SourceBlock, DestBlock*}
Values	{BlockValues *SourceBlock, DestBlock*}
Reformat	{BlockReformat *Block*}
Move Pages	{BlockMovePages *SrcPages, BeforePage*}
Insert Break	{InsertBreak}
Object Order	
Bring Forward	{FloatOrder.Forward}
Send Backward	{FloatOrder.Backward}
Bring To Front	{FloatOrder.ToFront}
Send to Back	{FloatOrder.ToBack}
Data	
Sort	
Block	{Sort.Block *Block*}
Reset	{Sort.Reset}
Column	
1st	{Sort.Key_1 *Block*}
2nd	{Sort.Key_2 *Block*}
3rd	{Sort.Key_3 *Block*}
4th	{Sort.Key_4 *Block*}
5th	{Sort.Key_5 *Block*}
Ascending	
1st	{Sort.Order_1 "Ascending" or "Descending"}
2nd	{Sort.Order_2 "Ascending" or "Descending"}
3rd	{Sort.Order_3 "Ascending" or "Descending"}
4th	{Sort.Order_4 "Ascending" or "Descending"}
5th	{Sort.Order_5 "Ascending" or "Descending"}
Data	
Numbers First	{Sort.Data "Numbers First"}
Labels First	{Sort.Data "Labels First"}
Labels	
Character Code	{Sort.Labels "Character Code"}
Dictionary	{Sort.Labels "Dictionary"}
OK	{Sort.Go}
Query	
Database Block	{Query.Database_Block *Block*}
Criteria Table	{Query.Criteria_Table *Block*}
Output Block	{Query.Output_Block *Block*}
Locate	{Query.Locate} enters FIND mode, {Query.EndLocate} returns the user to the dialog box
Delete	{Query.Delete}
Extract	{Query.Extract}
Field Names	{Query.Assign_Names}

Table A.1: Command equivalents by command (continued)

Command	Command equivalent
Extract Unique	{Query.Unique}
Reset	{Query.Reset}
Restrict Input	{RestrictInput.Enter *Block*} enters INPUT mode; {RestrictInput.Exit} returns Quattro Pro to READY mode
Parse	
Input	{Parse.Input *Block*}
Output	{Parse.Output *Block*}
Create	{Parse.Create}
Edit	{Parse.EditLine *NewEditLine*}
Reset	{Parse.Reset}
OK	{Parse.Go}
What-If	
What-If Table Type	
One-free variable	{WhatIf.One_Way}
Two-free variables	{WhatIf.Two_Way}
Data Table	{WhatIf.Block *Block*}
Input Cell	{WhatIf.Input_Cell_1 *Cell*}
Column Input Cell	{WhatIf.Input_Cell_1 *Cell*}
Row Input Cell	{WhatIf.Input_Cell_2 *Cell*}
Generate	{WhatIf.One_Way} or {WhatIf.Two_Way}
Reset	{WhatIf.Reset}
Frequency	
Value Blocks	{Frequency.Value_Block *Block*}
Bin Block	{Frequency.Bin_Block *Block*}
Reset	{Frequency.Reset}
OK	{Frequency.Go}
Database Desktop	{TableView}
Table Query	
Source Of Query	
Query In File	{TableQuery.FileQuery "Yes" or "No"}
Query In Block	{TableQuery.QueryBlock *Block*}
QBE File	{TableQuery.QueryFile *Filename*}
Destination	{TableQuery.Destination_Block *Block*}
OK	{TableQuery.Go}
Tools	
Macro	
Execute	{*MacroName*}
Record	None
Options	None
Debugger	{STEPON}
Disable Debugger	{STEPOFF}
Define Group	{Group.Define *GroupName, StartPage, EndPage*}
Delete	{Group.Delete *GroupName*}
	{Group.ResetNames} deletes all group names

Table A.1: Command equivalents by command (continued)

Command	Command equivalent
Combine	{FileCombine *FileName, <Blocks>, Operation*}
Operation	
Copy	{FileCombine *FileName, <Blocks>,* "Copy"}
Add	{FileCombine *FileName, <Blocks>,* "Add"}
Divide	{FileCombine *FileName, <Blocks>,* "Divide"}
Multiply	{FileCombine *FileName, <Blocks>,* "Multiply"}
Subtract	{FileCombine *FileName, <Blocks>,* "Subtract"}
Extract	{FileExtract "Formulas" or "Values", *Blocks, Filename*}
Option	
Formulas	{FileExtract "Formulas", *Blocks, Filename*}
Values	{FileExtract "Values", *Blocks, Filename*}
Import	
ASCII Text File	{FileImport *Filename,* "ASCII Text File"}
Comma & " Delimited File	{FileImport *Filename,* "Comma and "" Delimited File"}
Only Commas	{FileImport *Filename,* "Only Commas"}
Update Links	
Open Links	{Links.Open *LinkName*}
Refresh Links	{Links.Refresh *LinkName*}
Delete Links	{Links.Delete *LinkName*}
Change Link	{Links.Change *OldName, NewName*}
Advanced Math	
Regression	
Independent	{Regression.Independent *Block*}
Dependent	{Regression.Dependent *Block*}
Output	{Regression.Output *Block*}
Y Intercept	
Compute	{Regression.Y_Intercept "Compute"}
Zero	{Regression.Y_Intercept "Zero"}
Reset	{Regression.Reset}
OK	{Regression.Go}
Invert	
Source	{Invert.Source *Block*}
Destination	{Invert.Destination *Block*}
OK	{Invert.Go}
Multiply	
Matrix 1	{Multiply.Matrix_1 *Block*}
Matrix 2	{Multiply.Matrix_2 *Block*}
Destination	{Multiply.Destination *Block*}
OK	{Multiply.Go}
Optimizer	
Goal	
Solution Cell	{Optimizer.Solution_Cell *Cell*}
Max	{Optimizer.Solution_Goal *ObjectiveCell,* "Max"}
Min	{Optimizer.Solution_Goal *ObjectiveCell,* "Min"}

Table A.1: Command equivalents by command (continued)

Command	Command equivalent
None	{Optimizer.Solution_Goal *ObjectiveCell*, "None"}
Target Value	{Optimizer.Solution_Goal *ObjectiveCell*, "Target Value"} {Optimizer.Target_Value *Value*} sets the target value
Variable cells	{Optimizer.Variable_Cells *Blocks*}
Constraints	
Add	{Optimizer.Add *ConstraintString*}
Change	{Optimizer.Change *OldConstraint*, *NewConstraint*}
Delete	{Optimizer.Delete *ConstraintString*}
Reset	{Optimizer.Reset}
Options	
Max Time	{Optimizer.Max_Time *Value*}
Max Iterations	{Optimizer.Max_Iters *Value*}
Precision	{Optimizer.Precision *Value*}
Estimates	
Tangent	{Optimizer.Estimates "Tangent"}
Quadratic	{Optimizer.Estimates "Quadratic"}
Derivatives	
Forward	{Optimizer.Derivatives "Forward"}
Central	{Optimizer.Derivatives "Central"}
Search	
Newton	{Optimizer.Search "Newton"}
Conjugate	{Optimizer.Search "Conjugate"}
Show Iteration Results	{Optimizer.ShowIters 1 or 0}
Assume Linear	{Optimizer.Linear 1 or 0}
Load Model	Set cell to load from with {Optimizer.Model_Cell *Cell*}, then use {Optimizer.Load_Model}
Save Model	Set cell to save to with {Optimizer.Model_Cell *Cell*}, then use {Optimizer.Save_Model}
Reporting	
Answer Report Block	{Optimizer.Answer_Reporting *AnswerBlock*}
Detail Report Block	{Optimizer.Detail_Reporting *DetailBlock*}
Solve	{Optimizer.Solve}
Solve For	
Formula Cell	{SolveFor.Formula_Cell *Cell*}
Target Value	{SolveFor.Target_Value *Value*}
Variable Cell	{SolveFor.Variable_Cell *Cell*}
Max Iterations	{SolveFor.Max_Iters *Value*}
Accuracy	{SolveFor.Accuracy *Value*}
OK	{SolveFor.Go} {SolveFor.Reset} clears all Solve For settings
UI Builder	None

Table A.1: Command equivalents by command (continued)

Command	Command equivalent
Graph	
Type	{GraphSettings.Type *Type<,Class>*}
2-D	
Bar	{GraphSettings.Type "Bar,2-D"}
Variance	{GraphSettings.Type "Variance,2-D"}
Stacked Bar	{GraphSettings.Type "Stacked Bar,2-D"}
High Low	{GraphSettings.Type "HiLo,2-D"}
Line	{GraphSettings.Type "Line,2-D"}
XY	{GraphSettings.Type "XY,2-D"}
Area	{GraphSettings.Type "Area,2-D"}
Column	{GraphSettings.Type "Column"}
Pie	{GraphSettings.Type "Pie,2-D"}
3-D	
Bar	{GraphSettings.Type "3D Bar,3-D"}
Stacked Bar	{GraphSettings.Type "3D Stacked Bar,3-D"}
2.5-D Bar	{GraphSettings.Type "2DHalf Bar,3-D"}
Step	{GraphSettings.Type "3D Step,3-D"}
Unstacked Area	{GraphSettings.Type "3D Unstacked Area,3-D"}
Ribbon	{GraphSettings.Type "3D Ribbon,3-D"}
Area	{GraphSettings.Type "3D Area,3-D"}
Column	{GraphSettings.Type "3D Column,3-D"}
Pie	{GraphSettings.Type "3D Pie,3-D"}
Surface	{GraphSettings.Type "3D Surface,3-D"}
Contour	{GraphSettings.Type "3D Contour,3-D"}
Shaded Surface	{GraphSettings.Type "3D ShadedSurface,3-D"}
Rotate	
3D bar	{GraphSettings.Type "R3D bar,Rotate"}
2.5D Bar	{GraphSettings.Type "R2DHalf Bar,Rotate"}
2D Bar	{GraphSettings.Type "R2D Bar,Rotate"}
Area	{GraphSettings.Type "Rotated Area,Rotate"}
Line	{GraphSettings.Type "Rotated Line,Rotate"}
Combo	
Line - Bar	{GraphSettings.Type "Line_bar,Combo"}
Columns	{GraphSettings.Type "Multiple Columns,Combo"}
Area - Bar	{GraphSettings.Type "Area_bar,Combo"}
3D columns	{GraphSettings.Type "Multiple 3D columns,Combo"}
High Low - Bar	{GraphSettings.Type "Hilo_bar,Combo"}
Pies	{GraphSettings.Type "Multiple Pies,Combo"}
Bar	{GraphSettings.Type "Multiple Bar,Combo"}
3D Pies	{GraphSettings.Type "Multiple 3D Pies,Combo"}
Text	{GraphSettings.Type "Text,Text"}
Series	
Delete	{Series.Delete *SeriesNumber <,AndAllSeriesFollowing?>*}
Add	{Series.Insert *SeriesNumber, Block*}
1st...*n*th	{Series.Data_Range *SeriesNumber, Block <,CreateIfNonexistent?>*}

Table A.1: Command equivalents by command (continued)

Command	Command equivalent
X-Axis	{Series.Data_Range "XAxisLabelSeries", *Block*} sets the X-Axis series {Series.Label_Range *SeriesNumber, Block*<,CreateIfNonexistent>} sets the label series for a particular series
Legend	{Series.Data_Range "LegendSeries", *Block*} sets the legend series {Series.Legend *SeriesNumber, LegendText*} sets the legend text of a particular series
Reverse Series	{Series.Reverse_Series 1 or 0}
Row/column Swap	{Series.Swap_Row_Col 1 or 0}
Ok	{Series.Go}
Titles	{GraphSettings.Title *Main, Sub, X-Axis, Y-Axis, Y2-Axis*}
New	{GraphNew *Name*}
Edit	{GraphEdit *Name*}
Insert	{FLOATCREATE}
Delete	{GraphDelete *Name*}
Copy	{GraphCopy *GraphName, DestFile, Style?, Data?, Annotations?*}
View	{GraphView <*GraphName1, GraphName2,...*>}
Slide show	{Slide.Run *SlideName*}
Property	
Current Object	{SETPROPERTY *Property, Setting*}
Application	{Application.*Property*}
Display	{Application.Display *Clock, SpeedBar, InputLine, Status, BlockSyntax*}
Clock Display	{Application.Display.Clock_Display *String*}
None	{Application.Display.Clock_Display "None"}
Standard	{Application.Display.Clock_Display "Standard"}
International	{Application.Display.Clock_Display "International"}
Display Options	
Show SpeedBar	{Application.Display.Show_Toolbar "Yes" or "No"}
Show Input Line	{Application.Display.Show_InputLine "Yes" or "No"}
Show Status Line	{Application.Display.Show_StatusLine "Yes" or "No"}
3-D Syntax	{Application.Display.Range_Syntax "A..B:A1..B2" or "A:A1..B:B2"}
International	{Application.International *CurSym, Currency, Placement, Negative, Punctuation, DateFmt, TimeFmt, SortTable, LICS*}
Currency	
Currency	{Application.International.Currency "Windows Default" or "Quattro Pro/Windows"}
Currency Symbol	{Application.International.Currency_Symbol}
Placement	{Application.International.Placement "Prefix" or "Suffix"}
Negative Values	{Application.International.Negative "Signed" or "Parenthesis"}
Punctuation	{Application.International.Punctuation *String*}
Date Format	{Application.Date_Format *String*}

Table A.1: Command equivalents by command (continued)

Command	Command equivalent
Time Format	{Application.Time_Format *String*}
Language Conversion	{Application.International.Sort_Table *String*}
LICS	{Application.International.LICS_Conversion "Yes" or "No"}
Startup	{Application.Startup *StartDir, AutoFile, StartMacro, FileExt, UseBeep, UseUndo, CompatibleKeys*}
Directory	{Application.Startup.Startup_Directory *String*}
Autoload File	{Application.Startup.Autoload_File *String*}
Startup Macro	{Application.Startup.Startup_Macro *String*}
File Extension	{Application.Startup.File_Extension *String*}
Use Beep	{Application.Startup.Beep "Yes" or "No"}
Undo Enabled	{Application.Startup.Undo "Yes" or "No"}
Compatible Keys	{Application.Startup.Compatible_Keys "Yes" or "No"}
Macro	{Application.Macro *MacSuppress, KeyRead, SlashMen*}
Macro Suppress-Redraw	{Application.Macro.Macro_Redraw *MacSuppress*}
Both	{Application.Macro.Macro_Redraw "Both"}
Panel	{Application.Macro.Macro_Redraw "Panel"}
Window	{Application.Macro.Macro_Redraw "Window"}
None	{Application.Macro.Macro_Redraw "None"}
Key Reader	{Application.Macro.KeyReader "Yes" or "No"}
Slash Key	{Application.Macro.Slash_Key *MenuName*}
Title	{Application.Title *Title*}
SpeedBar	{Application.SpeedBar *SpeedBarName*}
Enable Inspection	{Application.Enable_Inspection "Yes" or "No"}
Active Notebook	{NoteBook.*Property*}
Recalc Settings	{NoteBook.Recalc_Settings *Mode, Iterations*}
Automatic	{NoteBook.Recalc_Settings "Automatic", *Iterations*}
Manual	{NoteBook.Recalc_Settings "Manual", *Iterations*}
Background	{NoteBook.Recalc_Settings "Background", *Iterations*}
Natural	{Notebook.Recalc_Settings "Natural", *Iterations*}
Column-wise	{NoteBook.Recalc_Settings "Column-wise", *Iterations*}
Row-wise	{NoteBook.Recalc_Settings "Row-wise", *Iterations*}
Zoom Factor	{NoteBook.Zoom_Factor 25-200}
Palette	
Color 1	{NoteBook.Palette.Color_1 "*Red,Green,Blue*"}
Color 2	{NoteBook.Palette.Color_2 "*Red,Green,Blue*"}
Color 3	{NoteBook.Palette.Color_3 "*Red,Green,Blue*"}
Color 4	{NoteBook.Palette.Color_4 "*Red,Green,Blue*"}
Color 5	{NoteBook.Palette.Color_5 "*Red,Green,Blue*"}
Color 6	{NoteBook.Palette.Color_6 "*Red,Green,Blue*"}
Color 7	{NoteBook.Palette.Color_7 "*Red,Green,Blue*"}
Color 8	{NoteBook.Palette.Color_8 "*Red,Green,Blue*"}
Color 9	{NoteBook.Palette.Color_9 "*Red,Green,Blue*"}
Color 10	{NoteBook.Palette.Color_10 "*Red,Green,Blue*"}
Color 11	{NoteBook.Palette.Color_11 "*Red,Green,Blue*"}

Table A.1: Command equivalents by command (continued)

Command	Command equivalent
Color 12	{NoteBook.Palette.Color_12 "*Red,Green,Blue*"}
Color 13	{NoteBook.Palette.Color_13 "*Red,Green,Blue*"}
Color 14	{NoteBook.Palette.Color_14 "*Red,Green,Blue*"}
Color 15	{NoteBook.Palette.Color_15 "*Red,Green,Blue*"}
Color 16	{NoteBook.Palette.Color_16 "*Red,Green,Blue*"}
Display	{NoteBook.Display *VertScroll, HorScroll, Tabs*}
Vertical Scroll Bar	{NoteBook.Display.Show_VerticalScroller "Yes" or "No"}
Horizontal Scroll Bar	{NoteBook.Display.Show_HorizontalScroller "Yes" or "No"}
Page Tabs	{NoteBook.Display.Show_Tabs "Yes" or "No"}
Macro Library	{NoteBook.Macro_Library "Yes" or "No"}
Active Page	{Page.*Property*}
Name	{Page.Name *NewName*}
Protection	{Page.Protection "Disable" or "Enable"}
Line Color	{Page.Line_Color 0-15}
Conditional Color	{Page.Conditional_Color *Enable, SmallVal, GreatVal, BelColor, NormalColor, AboveCol, ERRCol*}
Enable	{Page.Conditional_Color.Enable "Yes" or "No"}
Smallest Normal Value	{Page.Conditional_Color.Smallest_Normal_Value *Value*}
Greatest Normal Value	{Page.Conditional_Color.Greatest_Normal_Value *Value*}
Below Normal Color	{Page.Conditional_Color.Below_Normal_Color 0-15}
Normal Color	{Page.Conditional_Color.Normal_Color 0-15}
Above Normal Color	{Page.Conditional_Color.Above_Normal_Color 0-15}
ERR Color	{Page.Conditional_Color.ERR_Color 0-15}
Label Alignment	{Page.Label_Alignment *String*}
Left	{Page.Label_Alignment "Left"}
Right	{Page.Label_Alignment "Right"}
Center	{Page.Label_Alignment "Center"}
Display Zeros	{Page.Display_Zeros "Yes" or "No"}
Default Width	{Page.Default_Width *WidthInTwips*}
Borders	{Page.Borders *Row, Column*}
Row Borders	{Page.Borders.Row_Borders "Yes" or "No"}
Column Borders	{Page.Borders.Column_Borders "Yes" or "No"}
Grid Lines	{Page.Grid_Lines *Horizontal, Vertical*}
Horizontal	{Page.Grid_Lines.Horizontal "Yes" or "No"}
Vertical	{Page.Grid_Lines.Vertical "Yes" or "No"}
Graph Window	{GraphWindow.*Property*}
Aspect Ratio	{GraphWindow.Aspect_Ratio *Ratio*}
Floating Graph	{GraphWindow.Aspect_Ratio "Floating Graph"}
Screen Slide	{GraphWindow.Aspect_Ratio "Screen Slide"}
35mm Slide	{GraphWindow.Aspect_Ratio "35mm Slide"}
Printer Preview	{GraphWindow.Aspect_Ratio "Printer Preview"}
Full Extent	{GraphWindow.Aspect_Ratio "Full Extent"}
Grid	{GraphWindow.Grid *GridSize, DisplayGrid, SnapToGrid*}
Dialog	{DialogWindow.*Property*}
Dimension	{DialogWindow.Dimension *X, Y, Height, Width*}
X Pos	{DialogWindow.Dimension.X}
Y Pos	{DialogWindow.Dimension.Y}

Table A.1: Command equivalents by command (continued)

Command	Command equivalent
Height	{DialogWindow.Dimension.Height}
Width	{DialogWindow.Dimension.Width}
Title	{DialogWindow.Title}
Position Adjust	{DialogWindow.Position_Adjust *Use?, LeftRel, TopRel, RightRel, BottomRel, CenterHor, CenterVer*}
Grid Options	{DialogWindow.Grid_Options *GridSize, ShowGrid, EnableGrid*}
Name	{DialogWindow.Name *Name*}
Disabled	{DialogWindow.Disabled "Yes" or "No"}
Enabled	{DialogWindow.Enabled "Yes" or "No"}
Value	{DialogWindow.Value *String*}
Window	
New View	{WindowNewView}
Tile	{WindowTile}
Cascade	{WindowCascade}
Arrange Icons	{WindowArrIcon}
Hide	{WindowHide}
Show	{WindowShow *Name*}
Panes	{WindowPanes *SplitDirection, Synchronized?, Width, Height*}
Pane Options	
Horizontal	{WindowPanes "Horizontal", *Synchronized?, Width, Height*}
Vertical	{WindowPanes "Vertical", *Synchronized?, Width, Height*}
Clear	{WindowPanes "Clear", *Synchronized?, Width, Height*}
Synchronize	{WindowPanes *SplitDirection*, "Synch", *Width, Height*}
	{WindowPanes *SplitDirection*, "Unsynch", *Width, Height*}
Locked Titles	
Clear	{WindowTitles "Clear"}
Horizontal	{WindowTitles "Horizontal"}
Vertical	{WindowTitles "Vertical"}
Both	{WindowTitles "Both"}
1..*n* (list of open items)	{ACTIVATE *WindowName*} or {WINDOW*n*}
Help	None
Draw	
Group	{GroupObjects}
UnGroup	{UngroupObjects}
Bring Forward	{Order.Forward}
Send Backward	{Order.Backward}
Bring to Front	{Order.ToFront}
Send to Back	{Order.ToBack}
Align	
Left	{Align.Left}
Right	{Align.Right}
Horizontal Center	{Align.Horizontal_Center}
Top	{Align.Top}
Bottom	{Align.Bottom}
Vertical Center	{Align.Vertical_Center}

Table A.1: Command equivalents by command (continued)

Command	Command equivalent
Import	{ImportGraphic *Filename*}
Export	{ExportGraphic *Filename*, *<GrayScale>*, *<Compression>*}
Dialog	
Connect	None
Links	None
Align	
Left	{Align.Left}
Right	{Align.Right}
Horizontal Center	{Align.Horizontal_Center}
Top	{Align.Top}
Bottom	{Align.Bottom}
Vertical Center	{Align.Vertical_Center}
Horizontal Space	{Align.Horizontal_Space *Value*}
Vertical Space	{Align.Vertical_Space *Value*}
Resize to Same	{ResizeToSame}
Order	
Order Controls	{Controls.Order}
Order From	{Controls.OrderFrom}
Order Tab Controls	{Controls.OrderTab}
Order Tab From	{Controls.OrderTabFrom}
Bring Forward	{Order.Forward}
Send Backward	{Order.Backward}
Bring to Front	{Order.ToFront}
Send to Back	{Order.ToBack}
New SpeedBar	None
Open SpeedBar	None
Save SpeedBar	None
Save SpeedBar As	None
Close SpeedBar	None
OLE Object Inspectors	
Link settings	
Edit	{OLE.Edit}
Play	{OLE.Play}
Update now	{OLE.Update}
	{OLE.Update_All_Objects} updates all OLE objects
Unlink	{OLE.Unlink}
Change link	{OLE.Change_Link *FileName*}
Object settings	
Change to picture	{OLE.Change_To_Picture}
Edit	{OLE.Edit}
Play	{OLE.Play}

APPENDIX

B

Property reference

Quattro Pro has a variety of commands and @functions that affect property settings. This appendix lists the objects and properties you can manipulate using

- @PROPERTY
- {GETOBJECTPROPERTY}
- {GETPROPERTY}
- {SETOBJECTPROPERTY}
- {SETPROPERTY}
- link commands in a dialog box or menu command

The sections in this appendix list the various Quattro Pro objects and describe how to identify the objects in an @function or command.

Identifying objects

Entering a string into a command (or @function) that specifies the name of an object and property to manipulate is called *identifying* an object. The following table lists the ID syntax of Quattro Pro objects:

Table B.1: Identifying an object

Object	ID Syntax	Notes
Block (or Cell)	*[NBName]BlockAddress.Property*	*[NBName]* is optional.
Dialog box	*[NBName]DialogName:.Property*	*[NBName]* is optional.
Dialog control	*[NBName]DialogName:ObjName.Property*	*[NBName]* is optional. *DialogName*: isn't needed if the dialog box containing the control is active.
Dialog icon	*DialogName.Property*	Dialog icons can only be identified when the Graphs page is selected.
Floating object	*[NBName]Page:ObjName.Property*	*[NBName]* and *Page*: are optional.
Graph	*[NBName]GraphName.Property*	*[NBName]* is optional.
Graph icon	*GraphName.Property*	Graph icons can only be identified when the Graphs page is selected.
Graph object	*[NBName]GraphName:ObjName.Property*	*[NBName]* is optional. *GraphName*: isn't needed if the graph window containing the object is active.
Menu command	*Menupath.Property*	
Notebook	*[NBName].Property*	You can change properties of the active notebook using Active_Notebook.*Property*.
Page	Active_Page.*Property*	You can only read or set properties of the active page using this syntax.
Slideshow icon	*GraphName.Property*	Slideshow icons can only be identified when the Graphs page is selected.
SpeedBar	*[SpdBarName].Property*	
SpeedBar control	*[SpdBarName]ObjName.Property*	*[SpdBarName]* isn't needed if the SpeedBar containing the control is active.

In the previous table, *Property* is the name of the property setting to read or set. *ObjName* is the name of the object to manipulate. This name is stored one of the following properties: Name, Object Name, or Object ID.

The following two tables show some examples of how you can apply the information in the previous table to manipulate objects. The first table manipulates the Numeric Format property of the cell A:A23. The second manipulates the Disabled property of a bitmap button named Bitmap1 in a dialog box named Dialog1.

Table B.2: Manipulating a block property

Property Command	Notes
{GETOBJECTPROPERTY B:C32,"A:A23.Numeric_Format"}	The setting is stored in B:C32.
ON Init SET General TO A:A23.Numeric_Format	You can also use ID's with the SEND and RECEIVE link commands. This link command formats A:A23 as General.
@PROPERTY("A:A23.Numeric_Format")	
{SETOBJECTPROPERTY "A:A23.Numeric_Format","Currency,2"}	This command formats A:A23 as Currency (2 decimal places).

Table B.3: Manipulating a control property

Property Command	Notes
{GETOBJECTPROPERTY B:C32,"Dialog1:Bitmap1.Disabled"}	The setting is stored in B:C32.
ON Init SET Yes TO Dialog1:Bitmap1.Disabled	You can also use ID's with the SEND and RECEIVE link commands. This link command disables the button.
@PROPERTY("Dialog1:Bitmap1.Disabled")	
{SETOBJECTPROPERTY "Dialog1:Bitmap1.Disabled","No"}	This command enables the button.

{SETPROPERTY} and {GETPROPERTY} work on the selected object, and just take the name of the property to manipulate. The sections following this one discuss the syntax of each object in more detail.

Object precedence

If a situation arises where a property command could affect multiple objects (because they have the same name), the object highest on the following list is identified:

1. Dialog box
2. Graph
3. Floating object
4. Named block

When a Graphs page is active, you can't identify a dialog box or graph. The icon representing the dialog box or graph is identified instead.

Objects and properties

The property tables in this section are divided into three parts: the Object | Property column, the Argument column, and the Syntax column.

The Object | Property column lists the objects and their properties. The Argument column shows how to enter the property name in an @PROPERTY function or macro command. If a property name is followed by an asterisk (*), it's a hidden property and doesn't appear in the object's Object Inspector. If a property name is followed by `(R)`, you *cannot* set it with a macro command or link command; it is a read only property.

The Syntax column shows the layout of special property settings. If a syntax isn't listed, look at how the setting is changed in the object's Object Inspector menu. For example, because the Horizontal Scroll Bar option in a notebook Object Inspector is set with a check box, the property setting is entered as Yes or No; similarly, if a setting is changed by an edit field, you can use the text normally typed into that edit field.

Italicized items in the Syntax column describe the type of data returned; items in normal type are entered (when setting a property). If vertical bars (|) separate items, then only those items are returned or allowed in the property. For example, Both | Window | Panel | None means that the property setting is either Both, Window, Panel, or None; you can enter only one of these items to set the property. Another example is *Precision | Type,* which indicates that either the type or precision setting is listed in that position. Items in angle brackets (<>) are optional.

Common properties

Many objects contain the same type of property. This section lists some of the more common object properties.

Color

The Color property appears in many Object Inspector menus, but with a variation in title (Border_Color, Line_Color, Text_Color, and so on). The syntax of Color is:

Red,Green,Blue

Red is the amount of red in the color. *Green* is the amount of green in the color. *Blue* is the amount of blue in the color. Each of these components is an integer from 0 to 255.

When setting a color using an Object Inspector menu, you can click HSB or CMY (on a color control) to specify the color using a different color model (see Chapter 9 of the *User's Guide* for more information on color models). Using a different color model does not affect the syntax of the Color property; the Color property is always in Red,Green,Blue format.

You can also retrieve each of these color components individually. The following table lists the argument required to read or set an individual component of a Color property. *Item* is the word appearing in the Object Inspector (and property tables) that describes what color is being manipulated.

Table B.4
The Color properties

Property	**Argument**	**Syntax**
*Item*_Color	*Item*_Color	*Red,Green,Blue*
Red	*Item*_Color.Red	
Green	*Item*_Color.Green	
Blue	*Item*_Color.Blue	

Dimension

The Dimension property shows the precise size and position of an object relative to the window containing it. The syntax of Dimension is:

X,Y,Width,Height

X is the distance in pixels between the left edge of the object and the left side of the window. *Y* is the distance in pixels between the top edge of the object and the bottom edge of the window's title bar (or from the top of the graph background in the case of a drawn object). *Width* the width of the object, in pixels. *Height* is the height of the object, in pixels.

You can also retrieve each of these settings individually. The following table lists the argument required to read or set an individual option of the Dimension property.

Table B.5
The Dimension property

Property	**Argument**	**Syntax**
Dimension	Dimension	*X, Y, Width, Height*
X Pos	Dimension.X	
Y Pos	Dimension.Y	
Width	Dimension.Width	
Height	Dimension.Height	

Font

The Font property appears in many Object Inspector menus, but with a variation in title (Font, Label_Font, Text_Font). The syntax of Font is:

Typeface, PointSize, Bold, Italic, Underline, Strikeout

Typeface is the name of the typeface used. The list of typefaces available varies from system to system. *PointSize* is the size of the text, in points (a point is 1/72nd of an inch). *Bold, Italic, Underline,* and *Strikeout* each set one attribute of the text's appearance. Each is set to Yes or No.

You can also retrieve each of options individually. The following table lists the argument required to read or set an individual option of a Font property. *Item* is the word appearing in the Object Inspector (and property tables) that describes what font is being manipulated.

Table B.6
The Font properties

Property	Argument	Syntax
Font	Font	*Typeface, PointSize, Bold, Italic, Underline, Strikeout*
Typeface	Font.Typeface	
Point Size	Font.Point_Size	
Bold	Font.Bold	
Italic	Font.Italic	
Underline	Font.Underline	
Strikeout	Font.Strikeout	
Item Font	*Item*_Font	*Typeface, PointSize, Bold, Italic, Underline, Strikeout*
Typeface	*Item*_Font.Typeface	
Point Size	*Item*_Font.Point_Size	
Bold	*Item*_Font.Bold	
Italic	*Item*_Font.Italic	
Underline	*Item*_Font.Underline	
Strikeout	*Item*_Font.Strikeout	

Common graph object properties

The following table lists two properties that are found in many graph objects:

Table B.7
Common graph object properties

Property	Argument	Syntax
Fill Style	Fill_Style	None | Solid | Pattern | Wash | Bitmap, *Type*
None	Fill_Style	None,Empty
Solid	Fill_Style	Solid,Solid

Table B.7: Common graph object properties (continued)

Property	Argument	Syntax
Pattern	Fill_Style	Pattern,Solid
	Fill_Style	Pattern,Tight Dots
	Fill_Style	Pattern,Thick Stripes Down
	Fill_Style	Pattern,Thick Stripes Up
	Fill_Style	Pattern,Vertical Lines
	Fill_Style	Pattern,Horizontal Lines
	Fill_Style	Pattern,Horizontal Grid
	Fill_Style	Pattern,Hatch 1
	Fill_Style	Pattern,Diagonal 1
	Fill_Style	Pattern,Diagonal 2
	Fill_Style	Pattern,Vertical
	Fill_Style	Pattern,Horizontal
	Fill_Style	Pattern,Loose Dots
	Fill_Style	Pattern,Medium Dots
	Fill_Style	Pattern,Pepita
	Fill_Style	Pattern,Scales
	Fill_Style	Pattern,Diagonal Grid
	Fill_Style	Pattern,Hatch 2
	Fill_Style	Pattern,Fuzzy Stripes Down
	Fill_Style	Pattern,Weave
	Fill_Style	Pattern,Zig Zag
	Fill_Style	Pattern,Staggered Dashes
	Fill_Style	Pattern,Lattice
	Fill_Style	Pattern,Bricks
Wash	Fill_Style	Wash,Left to right
	Fill_Style	Wash,Right to left
	Fill_Style	Wash,Center to left and right
	Fill_Style	Wash,Top to bottom
	Fill_Style	Wash,Bottom to top
	Fill_Style	Wash,Center to top and bottom
Bitmap	Fill_Style	Bitmap,Crop to fit | Shrink to fit,*BitmapName*
Text Style	Text_Style	Solid | Wash | Bitmap, *WashType*, *BitmapName*,*Shadow*
Solid	Text_Style	Solid,None,,*Shadow*
Wash	Text_Style	Wash,Left to right,,*Shadow*
	Text_Style	Wash,Right to left,,*Shadow*
	Text_Style	Wash,Center to left and right,,*Shadow*
	Text_Style	Wash,Top to bottom,,*Shadow*
	Text_Style	Wash,Bottom to top,,*Shadow*
	Text_Style	Wash,Center to top and bottom,,*Shadow*
Bitmap	Text_Style	Bitmap,Crop to fit | Shrink to fit,*BitmapName*,*Shadow*

Active objects and menu commands

There are some situations where you want to view the property settings of an active object without using its exact name. You can use the names in the following table to identify these objects.

Table B.8
Identifying active objects

Active Object	Description
Active_Block	The selected block.
Active_Notebook	The active notebook window.
Active_Page	The active notebook page.

For example, the following formula returns the setting of the Macro Library property in the active notebook window:

@PROPERTY("Active_Notebook.Macro_Library")

You can also use {SETPROPERTY} and {GETPROPERTY} to manipulate to properties of selected objects (blocks, controls, graph objects, floating objects, and so on).

Each command on the menu bar has properties you can manipulate (see page 380 for details). To identify a menu command, enter its path separated by forward slashes (/). Don't include ellipses (...). For example, the following formula returns whether Edit | Paste Format is grayed on the menu:

@PROPERTY("/Edit/Paste Special.Grayed")

The following table lists some special ways to identify menu commands.

Table B.9
Identifying menu commands

ID Syntax	Description
/<-	The first item on the menu bar. You can include this name at the end of a menu path (for example, `/File/<-` identifies the first item on the File menu).
/->	The last item on the menu bar. You can include this name at the end of a menu path (for example, `/Tools/Macro/->` identifies the last item on the Tools | Macro menu).
/*n*	The *n*th item on the menu bar. You can include this name at the end of a menu path (for example, `/Help/2` identifies the second item on the Help menu).

The following table lists the properties of active objects and menu commands.

Table B.10: Active objects and menu commands

Object|Property	Argument	Syntax
Active Block	see Block on page 426 for a list of block properties	
Active Notebook	see Notebook on page 427 for a list of notebook properties	
Active Page		
Name	Name	
Protection	Protection	Disable | Enable
Line Color	Line_Color	0-15
Conditional Color	Conditional_Color	*Enable, SmallVal, GreatVal, BelColor, Normal, AboveCol, ERRCol*
Enable	Conditional_Color.Enable	
Smallest Normal Value	Conditional_Color.Smallest_Normal_Value	
Greatest Normal Value	Conditional_Color.Greatest_Normal_Value	
Note: Each color property listed below has 0-15 as a setting.		
Below Normal Color	Conditional_Color.Below_Normal_Color	
Normal Color	Conditional_Color.Normal_Color	
Above Normal Color	Conditional_Color.Above_Normal_Color	
ERR Color	Conditional_Color.ERR_Color	
Label Alignment	Label_Alignment	Left | Right | Center
Display Zeros	Display_Zeros	
Default Width	Default_Width	
Borders	Borders	*Row, Column*
Row Borders	Borders.Row_Borders	
Column Borders	Borders.Column_Borders	
Grid Lines	Grid_Lines	*Horizontal, Vertical*
Horizontal	Grid_Lines.Horizontal	
Vertical	Grid_Lines.Vertical	
Menu Command		
Title *	Title	
Checked *	Checked	
HotKey *	HotKey	
Help Line *	Help_Line	
Depend On *	Depend_On	*Desktop, NoteWin, GraphWin, DiaWin, EditWin, GraphsPage*
Enabled *	Enabled	
Grayed *	Grayed	
Disabled *	Disabled	
Hidden *	Hidden	
Show *	Show	

Dialog box objects

To identify a dialog box (not the objects in it), use its name followed by a colon. For example, @PROPERTY("Dialog1:.Title") returns the Title property of the dialog box Dialog1. To see the property settings of a dialog box outside of the active notebook, enter the notebook name in brackets before the dialog box name:

[NOTEBK1.WB1]Dialog1

You can use the following syntax to identify objects in a dialog box:

[Notebook]DialogName:ObjectName.Property

Notebook is the name of the notebook containing the dialog box; it's optional. *DialogName* is the name of the dialog box containing the object (omit *DialogName*: if the dialog window containing the object is active). *ObjectName* is either the object ID number of the object (found in its Object ID property) or the name of the object (found in its Name property). *Property* is one of the strings listed in the Argument column of the next table. For example, the following formula returns the Dimension property of the object named Button1 in the dialog box Dialog1:

@PROPERTY("Dialog1:Button1.Dimension")

You can use a similar syntax to identify SpeedBars, but the name of the SpeedBar must be in brackets ([]). For example, the following macro disables the SpeedBar QUIKSAVE.BAR:

{SETOBJECTPROPERTY "[QUIKSAVE.BAR].Disabled","Yes"}

You can use the following syntax to identify objects in a SpeedBar:

[SpdBarName]ObjectName.Property

SpdBarName is the name of the SpeedBar containing the object (omit *[SpdBarName]* if the SpeedBar containing the object is active). *ObjectName* is either the object ID number of the object (found in its Object ID property) or the name of the object (found in its Name property). *Property* is one of the strings listed in the Argument column of the next table. For example, the following formula returns the Dimension property of the object named Button1 in the SpeedBar QUIKSAVE.BAR:

@PROPERTY("[QUIKSAVE.BAR]Button1.Dimension")

The following table lists each dialog object, the dialog box as an object, and the SpeedBar as an object.

Table B.11: Dialog objects and properties

Object|Property	Argument	Syntax
Bitmap Button		
Bitmap	Bitmap	
Label Text	Label_Text	
Text Draw Flags	Text_Draw_Flags	*Apply, HorCenter, AlignRight(Yes)orLeft(No), VertCenter, AlignBottom(Yes)orTop(No), WordBreak, SingleLine*
Default Button	Default_Button	
Button Type	Button_Type	Push Button | Radio Button | Check Box | OK Exit Button | Cancel Exit Button
Dimension	Dimension	see page 403
Hidden	Hidden	
Object ID (R)	Object_ID	
Position Adjust	Position_Adjust	*Depend, LeftRel, TopRel, RightRel, BottomRel, CenterHor, CenterVer*
Name	Name	
Help Line	Help_Line	
Grayed	Grayed	
Disabled	Disabled	
Depend On	Depend_On	*Desktop, NoteWin, GraphWin, DiaWin, EditWin, GraphsPage*
Tab Stop	Tab_Stop	
Enabled *	Enabled	
Show *	Show	
Value *	Value	
Button		
Label Text	Label_Text	
Text Draw Flags	Text_Draw_Flags	*Apply, HorCenter, AlignRight(Yes)orLeft(No), VertCenter, AlignBottom(Yes)orTop(No), WordBreak, SingleLine*
Default Button	Default_Button	
Button Type	Button_Type	Push Button | Radio Button | Check Box | OK Exit Button | Cancel Exit Button
Dimension	Dimension	see page 403
Hidden	Hidden	
Object ID (R)	Object_ID	
Position Adjust	Position_Adjust	*Depend, LeftRel, TopRel, RightRel, BottomRel, CenterHor, CenterVer*
Name	Name	
Help Line	Help_Line	
Grayed	Grayed	
Disabled	Disabled	
Depend On	Depend_On	*Desktop, NoteWin, GraphWin, DiaWin, EditWin, GraphsPage*
Tab Stop	Tab_Stop	
Enabled *	Enabled	

Table B.11: Dialog objects and properties (continued)

Object|Property	Argument	Syntax
Show *	Show	
Value *	Value	
Check Box		
Draw to Right	Draw_to_right	
Process Value	Process_Value	
Label Text	Label_Text	
Text Draw Flags	Text_Draw_Flags	*Apply, HorCenter, AlignRight(Yes)orLeft(No), VertCenter, AlignBottom(Yes)orTop(No), WordBreak, SingleLine*
Button Type	Button_Type	Push Button | Radio Button | Check Box | OK Exit Button | Cancel Exit Button
Dimension	Dimension	see page 403
Hidden	Hidden	
Object ID (R)	Object_ID	
Position Adjust	Position_Adjust	*Depend, LeftRel, TopRel, RightRel, BottomRel, CenterHor, CenterVer*
Name	Name	
Help Line	Help_Line	
Grayed	Grayed	
Disabled	Disabled	
Depend On	Depend_On	*Desktop, NoteWin, GraphWin, DiaWin, EditWin, GraphsPage*
Tab Stop	Tab_Stop	
Enabled *	Enabled	
Show *	Show	
Value *	Value	
Color Control		
Dimension	Dimension	see page 403
Hidden	Hidden	
Object ID (R)	Object_ID	
Position Adjust	Position_Adjust	*Depend, LeftRel, TopRel, RightRel, BottomRel, CenterHor, CenterVer*
Name	Name	
Help Line	Help_Line	
Grayed	Grayed	
Disabled	Disabled	
Process Value	Process_Value	
Tab Stop	Tab_Stop	
Enabled *	Enabled	
Show *	Show	
Value *	Value	*Red,Green,Blue*
Combo Box		
List	List	
Allow Point Mode	Allow_Point_Mode	
Add Down Button	Add_Down_Button	
History List	History_List	
List Length	List_Length	
Terminate Dialog	Terminate_Dialog	

Table B.11: Dialog objects and properties (continued)

Object|Property	Argument	Syntax
Edit Disabled	Edit_Disabled	
Edit Length	Edit_Length	
Ordered	Ordered	
Dimension	Dimension	see page 403
Hidden	Hidden	
Object ID (R)	Object_ID	
Position Adjust	Position_Adjust	*Depend, LeftRel, TopRel, RightRel, BottomRel, CenterHor, CenterVer*
Name	Name	
Help Line	Help_Line	
Grayed	Grayed	
Disabled	Disabled	
Depend On	Depend_On	*Desktop, NoteWin, GraphWin, DiaWin, EditWin, GraphsPage*
Process Value	Process_Value	
Tab Stop	Tab_Stop	
Enabled *	Enabled	
Show *	Show	
Value *	Value	
Dialog Box (Dialog)		
Dimension	Dimension	see page 403
Title	Title	
Position Adjust	Position_Adjust	*Depend, LeftRel, TopRel, RightRel, BottomRel, CenterHor, CenterVer*
Grid Options	Grid_Options	*Gridsize, GridShown, GridEnabled*
Name	Name	
Disabled	Disabled	
Enabled *	Enabled	
Value *	Value	
Edit Field		
Field Type	Field_Type	Integer | String | Real | Range | Hidden
Edit Length	Edit_Length	
Allow Point Mode	Allow_Point_Mode	
Show Frame	Show_Frame	
Terminate Dialog	Terminate_Dialog	
Convert Text	Convert_Text	
Dimension	Dimension	see page 403
Hidden	Hidden	
Object ID (R)	Object_ID	
Position Adjust	Position_Adjust	*Depend, LeftRel, TopRel, RightRel, BottomRel, CenterHor, CenterVer*
Name	Name	
Help Line	Help_Line	
Grayed	Grayed	
Disabled	Disabled	
Depend On	Depend_On	*Desktop, NoteWin, GraphWin, DiaWin, EditWin, GraphsPage*

Table B.11: Dialog objects and properties (continued)

Object|Property	Argument	Syntax
Process Value	Process_Value	
Tab Stop	Tab_Stop	
Enabled *	Enabled	
Show *	Show	
Value *	Value	
Edit Integer		
Minimum	Minimum	
Maximum	Maximum	
Default	Default	
Field Type	Field_Type	Integer | String | Real | Range | Hidden
Edit Length	Edit_Length	
Show Frame	Show_Frame	
Terminate Dialog	Terminate_Dialog	
Dimension	Dimension	see page 403
Hidden	Hidden	
Object ID (R)	Object_ID	
Position Adjust	Position_Adjust	*Depend, LeftRel, TopRel, RightRel, BottomRel, CenterHor, CenterVer*
Name	Name	
Help Line	Help_Line	
Grayed	Grayed	
Disabled	Disabled	
Depend On	Depend_On	*Desktop, NoteWin, GraphWin, DiaWin, EditWin, GraphsPage*
Process Value	Process_Value	
Tab Stop	Tab_Stop	
Enabled *	Enabled	
Show *	Show	
Value *	Value	
File Control		
Dimension	Dimension	see page 403
Hidden	Hidden	
Object ID (R)	Object_ID	
Attach Child	Attach_Child	
Position Adjust	Position_Adjust	*Depend, LeftRel, TopRel, RightRel, BottomRel, CenterHor, CenterVer*
Name	Name	
Help Line	Help_Line	
Grayed	Grayed	
Disabled	Disabled	
Depend On	Depend_On	*Desktop, NoteWin, GraphWin, DiaWin, EditWin, GraphsPage*
Process Value	Process_Value	
Tab Stop	Tab_Stop	
Enabled *	Enabled	
Show *	Show	
Value *	Value	

Table B.11: Dialog objects and properties (continued)

Object|Property	Argument	Syntax
Group Box		
Group Text	Group_Text	
Selected	Selected	
Dimension	Dimension	see page 403
Hidden	Hidden	
Object ID (R)	Object_ID	
Attach Child	Attach_Child	
Position Adjust	Position_Adjust	*Depend, LeftRel, TopRel, RightRel, BottomRel, CenterHor, CenterVer*
Name	Name	
Disabled	Disabled	
Process Value	Process_Value	
Enabled *	Enabled	
Show *	Show	
Value *	Value	
Horizontal Scroller (HScrollBar)		
Parameters	Parameters	*Min, Max, Line, Page, Time*
Dimension	Dimension	see page 403
Hidden	Hidden	
Object ID (R)	Object_ID	
Position Adjust	Position_Adjust	*Depend, LeftRel, TopRel, RightRel, BottomRel, CenterHor, CenterVer*
Name	Name	
Help Line	Help_Line	
Grayed	Grayed	
Disabled	Disabled	
Depend On	Depend_On	*Desktop, NoteWin, GraphWin, DiaWin, EditWin, GraphsPage*
Process Value	Process_Value	
Tab Stop	Tab_Stop	
Enabled *	Enabled	
Show *	Show	
Value *	Value	
Label		
Label Text	Label_Text	
Label Font	Label_Font	see page 404
Text Draw Flags	Text_Draw_Flags	*Apply, HorCenter, AlignRight(Yes)orLeft(No), VertCenter, AlignBottom(Yes)orTop(No), WordBreak, SingleLine*
Dimension	Dimension	see page 403
Hidden	Hidden	
Object ID (R)	Object_ID	
Position Adjust	Position_Adjust	*Depend, LeftRel, TopRel, RightRel, BottomRel, CenterHor, CenterVer*
Grayed	Grayed	
Name	Name	
Disabled	Disabled	

Table B.11: Dialog objects and properties (continued)

Object|Property	Argument	Syntax
Depend On	Depend_On	*Desktop, NoteWin, GraphWin, DiaWin, EditWin, GraphsPage*
Process Value	Process_Value	
Enabled *	Enabled	
Show *	Show	
Value *	Value	
List Box		
List	List	
Ordered	Ordered	
Number of Columns	Number_of_Columns	
Selection Text	Selection_Text	
Selected	Selected	
Dimension	Dimension	see page 403
Hidden	Hidden	
Object ID (R)	Object_ID	
Attach Child	Attach_Child	
Position Adjust	Position_Adjust	*Depend, LeftRel, TopRel, RightRel, BottomRel, CenterHor, CenterVer*
Name	Name	
Help Line	Help_Line	
Grayed	Grayed	
Disabled	Disabled	
Depend On	Depend_On	*Desktop, NoteWin, GraphWin, DiaWin, EditWin, GraphsPage*
Process Value	Process_Value	
Tab Stop	Tab_Stop	
Enabled *	Enabled	
Show *	Show	
Value *	Value	
Pick List		
Title	Title	
Selected	Selected	
Resize	Resize	
Dimension	Dimension	see page 403
Hidden	Hidden	
Object ID (R)	Object_ID	
Attach Child	Attach_Child	
Position Adjust	Position_Adjust	*Depend, LeftRel, TopRel, RightRel, BottomRel, CenterHor, CenterVer*
Name	Name	
Help Line	Help_Line	
Grayed	Grayed	
Disabled	Disabled	
Depend On	Depend_On	*Desktop, NoteWin, GraphWin, DiaWin, EditWin, GraphsPage*
Process Value	Process_Value	
Tab Stop	Tab_Stop	

Table B.11: Dialog objects and properties (continued)

Object|Property	Argument	Syntax
Enabled *	Enabled	
Show *	Show	
Value *	Value	
Radio Button		
Draw to Right	Draw_to_right	
Process Value	Process_Value	
Label Text	Label_Text	
Text Draw Flags	Text_Draw_Flags	*Apply, HorCenter, AlignRight(Yes)orLeft(No), VertCenter, AlignBottom(Yes)orTop(No), WordBreak, SingleLine*
Button Type	Button_Type	Push Button | Radio Button | Check Box | OK Exit Button | Cancel Exit Button
Dimension	Dimension	see page 403
Hidden	Hidden	
Object ID (R)	Object_ID	
Position Adjust	Position_Adjust	*Depend, LeftRel, TopRel, RightRel, BottomRel, CenterHor, CenterVer*
Name	Name	
Help Line	Help_Line	
Grayed	Grayed	
Disabled	Disabled	
Depend On	Depend_On	*Desktop, NoteWin, GraphWin, DiaWin, EditWin, GraphsPage*
Tab Stop	Tab_Stop	
Enabled *	Enabled	
Show *	Show	
Value *	Value	
Rectangle		
Rectangle Style	Rectangle_Style	Plain | Framed | Beveled Out | Beveled In | Transparent
Fill Color	Fill_Color	see page 402
Frame Color	Frame_Color	see page 402
Dimension	Dimension	see page 403
Hidden	Hidden	
Object ID (R)	Object_ID	
Attach Child	Attach_Child	
Position Adjust	Position_Adjust	*Depend, LeftRel, TopRel, RightRel, BottomRel, CenterHor, CenterVer*
Name	Name	
Disabled	Disabled	
Depend On	Depend_On	*Desktop, NoteWin, GraphWin, DiaWin, EditWin, GraphsPage*
Enabled *	Enabled	
Show *	Show	
ScrollBar	see Vertical Scroller	
SpeedBar	see Dialog Box	

Table B.11: Dialog objects and properties (continued)

Object|Property	Argument	Syntax
Spin Control		
Edit Length	Edit_Length	
Show Frame	Show_Frame	
Minimum	Minimum	
Maximum	Maximum	
Default	Default	
Dimension	Dimension	see page 403
Hidden	Hidden	
Object ID (R)	Object_ID	
Attach Child	Attach_Child	
Position Adjust	Position_Adjust	*Depend, LeftRel, TopRel, RightRel, BottomRel, CenterHor, CenterVer*
Name	Name	
Help Line	Help_Line	
Grayed	Grayed	
Disabled	Disabled	
Depend On	Depend_On	*Desktop, NoteWin, GraphWin, DiaWin, EditWin, GraphsPage*
Process Value	Process_Value	
Tab Stop	Tab_Stop	
Enabled *	Enabled	
Show *	Show	
Value *	Value	
Timer Control (TimerCtrl)		
Show Time	Show_Time	
Interval in Units	Interval_In_Units	
Units in Milliseconds	Units in Milliseconds	
Current Time (R)	Current_Time	*Hour,Minute,Second*
<u>Hour</u>	Current_Time.Hour	
<u>Minute</u>	Current_Time.Minute	
<u>Second</u>	Current_Time.Second	
Timer On	Timer_On	
Alarm Time	Alarm_Time	*Hour,Minute,Second*
<u>Hour</u>	Alarm_Time.Hour	
<u>Minute</u>	Alarm_Time.Minute	
<u>Second</u>	Alarm_Time.Second	
Alarm On	Alarm_On	
Dimension	Dimension	see page 403
Hidden	Hidden	
Object ID (R)	Object_ID	
Attach Child	Attach_Child	
Position Adjust	Position_Adjust	*Depend, LeftRel, TopRel, RightRel, BottomRel, CenterHor, CenterVer*
Name	Name	
Depend On	Depend_On	*Desktop, NoteWin, GraphWin, DiaWin, EditWin, GraphsPage*

Table B.11: Dialog objects and properties (continued)

Object|Property	Argument	Syntax
Process Value	Process_Value	
Show *	Show	
Value *	Value	
Vertical Scroller (ScrollBar)		
Parameters	Parameters	*Min, Max, Line, Page, Time*
Dimension	Dimension	see page 403
Hidden	Hidden	
Object ID (R)	Object_ID	
Position Adjust	Position_Adjust	*Depend, LeftRel, TopRel, RightRel, BottomRel, CenterHor, CenterVer*
Name	Name	
Help Line	Help_Line	
Grayed	Grayed	
Disabled	Disabled	
Depend On	Depend_On	*Desktop, NoteWin, GraphWin, DiaWin, EditWin, GraphsPage*
Process Value	Process_Value	
Tab Stop	Tab_Stop	
Enabled *	Enabled	
Show *	Show	
Value *	Value	

Graph objects (drawn)

Drawn objects in a graph are created by the graph SpeedBar; see the next section for a list of fixed graph objects like the x-axis. To identify a drawn object in a graph, use the following syntax:

[*Notebook*]*GraphName*:*ObjectName*.*Property*

Some properties of a drawn object only appear in an Object Inspector menu when Quattro Pro is loaded with the /D command line switch.

[*Notebook*] is the name of the notebook containing the graph; it's optional. *GraphName* is the name of the graph containing the object (omit *GraphName*: if the graph window containing the object is active). *ObjectName* is either the object ID number of the object (found in the Object ID property) or the name of the object (found in the Name property). *Property* is one of the strings listed in the Argument column of the next table. For example, the following formula returns the Line Style property of the object named ProfitCallout in the graph YTD1:

```
@PROPERTY("YTD1:ProfitCallout.Line_Style")
```

The following table lists each drawn object. See the next section for a list of fixed graph objects like the x-axis.

Table B.12: Graph objects (drawn)

Object|Property	Argument	Syntax
Arrow		
Fill Color	Fill_Color	see page 402
Bkg Color	Bkg_Color	see page 402
Fill Style	Fill_Style	see page 404
Border Color	Border_Color	see page 402
Border Style	Border_Style	S0W1 | S0W2 | S0W3 | S0W4 | S1W1 | S2W1 | S3W1 | S4W1 | S5W1
Name	Name	
Dimension	Dimension	see page 403
Object ID * (R)	Object_ID	
Box		
Graph Button	Graph_Button	*GotoSlide?*, *SlideName*, *Effect*, *Duration*, Slow | Med | Fast, *Overlay?*, *RunMacro?*, *MacroText*
Alignment	Alignment	Left | Right | Center, *WordWrap*, *TabStops*
Box Type	Box_Type	No box | Single outline | Rounded corners | Double outline | Thick outline 1 | Thick rounded corners | Three dimensional | Thick outline 2 | Bevel out | Shadowed | Thick outline 3 | Bevel in
Fill Color	Fill_Color	see page 402
Bkg Color	Bkg_Color	see page 402
Fill Style	Fill_Style	see page 404
Border Color	Border_Color	see page 402
Name	Name	
Dimension	Dimension	see page 403
Object ID * (R)	Object_ID	
Ellipse		
Fill Color	Fill_Color	see page 402
Bkg Color	Bkg_Color	see page 402
Fill Style	Fill_Style	see page 404
Border Color	Border_Color	see page 402
Border Style	Border_Style	S0W1 | S0W2 | S0W3 | S0W4 | S1W1 | S2W1 | S3W1 | S4W1 | S5W1
Name	Name	
Dimension	Dimension	see page 403
Object ID * (R)	Object_ID	
Freehand Polygon		
Fill Color	Fill_Color	see page 402
Bkg Color	Bkg_Color	see page 402
Fill Style	Fill_Style	see page 404
Border Color	Border_Color	see page 402
Border Style	Border_Style	S0W1 | S0W2 | S0W3 | S0W4 | S1W1 | S2W1 | S3W1 | S4W1 | S5W1
Name	Name	
Dimension	Dimension	see page 403
Object ID * (R)	Object_ID	

Table B.12: Graph objects (drawn) (continued)

Object|Property	Argument	Syntax
Freehand Polyline		
Line Color	Line_Color	see page 402
Line Style	Line_Style	S0W1 | S0W2 | S0W3 | S0W4 | S1W1 | S2W1 | S3W1 | S4W1 | S5W1
Name	Name	
Dimension	Dimension	see page 403
Object ID * (R)	Object_ID	
Line		
Line Color	Line_Color	see page 402
Line Style	Line_Style	S0W1 | S0W2 | S0W3 | S0W4 | S1W1 | S2W1 | S3W1 | S4W1 | S5W1
Name	Name	
Dimension	Dimension	see page 403
Object ID * (R)	Object_ID	
Polygon		
Fill Color	Fill_Color	see page 402
Bkg Color	Bkg_Color	see page 402
Fill Style	Fill_Style	see page 404
Border Color	Border_Color	see page 402
Border Style	Border_Style	S0W1 | S0W2 | S0W3 | S0W4 | S1W1 | S2W1 | S3W1 | S4W1 | S5W1
Name	Name	
Dimension	Dimension	see page 403
Object ID * (R)	Object_ID	
Polyline		
Line Color	Line_Color	see page 402
Line Style	Line_Style	S0W1 | S0W2 | S0W3 | S0W4 | S1W1 | S2W1 | S3W1 | S4W1 | S5W1
Name	Name	
Dimension	Dimension	see page 403
Object ID * (R)	Object_ID	
Rectangle		
Fill Color	Fill_Color	see page 402
Bkg Color	Bkg_Color	see page 402
Fill Style	Fill_Style	see page 404
Border Color	Border_Color	see page 402
Border Style	Border_Style	S0W1 | S0W2 | S0W3 | S0W4 | S1W1 | S2W1 | S3W1 | S4W1 | S5W1
Name	Name	
Dimension	Dimension	see page 403
Object ID * (R)	Object_ID	
Rounded Rectangle		
Fill Color	Fill_Color	see page 402
Bkg Color	Bkg_Color	see page 402
Fill Style	Fill_Style	see page 404
Border Color	Border_Color	see page 402

Table B.12: Graph objects (drawn) (continued)

Object|Property	Argument	Syntax
Border Style	Border_Style	S0W1 | S0W2 | S0W3 | S0W4 | S1W1 | S2W1 | S3W1 | S4W1 | S5W1
Name	Name	
Dimension	Dimension	see page 403
Object ID * (R)	Object_ID	
Text		
Text Color	Text_Color	see page 402
Text Bkg Color	Text_Bkg_Color	see page 402
Text Font	Text_Font	see page 404
Text Style	Text_Style	see page 405
Value	Value	Contents of the text box

Graph objects (fixed)

To set a property of the graph (not the objects in it), use its name followed by a period. For example, the following formula returns the Aspect Ratio property of the graph Graph1.

@PROPERTY("Graph1.Aspect_Ratio")

To identify a fixed object in a graph (as opposed to an object created with the graph SpeedBar; see the previous section for a list of drawn graph objects) use the following syntax:

[Notebook]GraphName:ObjectName.Property

Notebook is the name of the notebook containing the graph; it's optional. *GraphName* is the name of the graph containing the object (omit *GraphName* if the graph window containing the object is active). *ObjectName* is set to one of the names listed in the following table.

Table B.13
Fixed object names

Object|Property	Description
G$Base	Base of a 3-D graph
G$Graph	Background of the graph window
G$LeftWall	Left wall of a 3-D graph grid
G$Legend	Graph legend
G$Pane	Pane of a 2-D graph
G$RightWall	Back wall of a 3-D graph grid
G$Series[*x*,*y*]	*y*th data point of the *x*th series in the graph
G$SeriesLabel	Series labels
G$Title	Title, subtitle, and title box of the graph
G$X1Axis	x-axis
G$X1Title	x-axis title
G$Y1Axis	Primary y-axis

Table B.13: Fixed object names (continued)

Object|Property	Description
G$Y1Title	Primary y-axis title
G$Y2Axis	Secondary y-axis
G$Y2Title	Secondary y-axis title

Property is one of the strings listed in the Argument column of the next table. For example, the following formula returns the color of the third series of the graph PROFIT.

@PROPERTY("PROFIT:G$Series[3,1].Fill_Color")

The following table lists each fixed object and the properties of the graph itself. See the previous section for a list of drawn graph objects (such as arrows).

Table B.14: Graph objects (fixed)

Object|Property	Argument	Syntax
Area Fill		
Fill Color	Fill_Color	see page 402
Bkg Color	Bkg_Color	see page 402
Fill Style	Fill_Style	see page 404
Border Color	Border_Color	see page 402
Border Style	Border_Style	S0W1 | S0W2 | S0W3 | S0W4 | S1W1 | S2W1 | S3W1 | S4W1 | S5W1
Axis Title		
Title	Title	
Text Color	Text_Color	see page 402
Text Bkg Color	Text_Bkg_Color	see page 402
Text Font	Text_Font	see page 404
Text Style	Text_Style	see page 405
Bar Series		
Series Options	Series_Options	*DataBlock*, *LabelBlock*, *Legend*, Bar | Line | Area | Default, Primary | Secondary
Bar Options	Bar_Options	*Width%*, *Margin%*, No | Partial | Full
Fill Color	Fill_Color	see page 402
Bkg Color	Bkg_Color	see page 402
Fill Style	Fill_Style	see page 404
Border Color	Border_Color	see page 402
Border Style	Border_Style	S0W1 | S0W2 | S0W3 | S0W4 | S1W1 | S2W1 | S3W1 | S4W1 | S5W1
Line Series		
Series Options	Series_Options	*DataBlock*, *LabelBlock*, *Legend*, Bar | Line | Area | Default, Primary | Secondary
Marker Style	Marker_Style	M00 | M01 | M02 | M03 | M04 | M05 | M06 | M07 | M08 | M09 | M10 | M11 | M12 | M13 | M14 | M15, *MarkerWeight*

Table B.14: Graph objects (fixed) (continued)

Object|Property	Argument	Syntax
Fill Color	Fill_Color	see page 402
Bkg Color	Bkg_Color	see page 402
Fill Style	Fill_Style	see page 404
Line Color	Line_Color	see page 402
Line Style	Line_Style	S0W1 | S0W2 | S0W3 | S0W4 | S1W1 | S2W1 | S3W1 | S4W1 | S5W1
Area Series		
Series Options	Series_Options	*DataBlock, LabelBlock, Legend*
Fill Color	Fill_Color	see page 402
Bkg Color	Bkg_Color	see page 402
Fill Style	Fill_Style	see page 404
Border Color	Border_Color	see page 402
Border Style	Border_Style	S0W1 | S0W2 | S0W3 | S0W4 | S1W1 | S2W1 | S3W1 | S4W1 | S5W1
Column Graph		
Label Options	Label_Options	*LabelSeries*, Currency | Value | Percent | None, *ShowTick*
Text Color	Text_Color	see page 402
Text Bkg Color	Text_Bkg_Color	see page 402
Text Font	Text_Font	see page 404
Text Style	Text_Style	see page 405
Fill Color	Fill_Color	see page 402
Bkg Color	Bkg_Color	see page 402
Fill Style	Fill_Style	see page 404
Border Color	Border_Color	see page 402
Border Style	Border_Style	S0W1 | S0W2 | S0W3 | S0W4 | S1W1 | S2W1 | S3W1 | S4W1 | S5W1
Graph	see Graph Window and Graph Setup and Background for graph properties	
Graph Pane		
Border Options	Border_Options	*Left, Top, Right, Bottom*
Fill Color	Fill_Color	see page 402
Bkg Color	Bkg_Color	see page 402
Fill Style	Fill_Style	see page 404
Border Color	Border_Color	see page 402
Border Style	Border_Style	S0W1 | S0W2 | S0W3 | S0W4 | S1W1 | S2W1 | S3W1 | S4W1 | S5W1
Graph Setup And Background		
Graph Type		
2-D	Graph_Type	Bar | Variance | Stacked Bar | HiLo | Line | XY | Area | Column | Pie
3-D	Graph_Type	3D Bar | 3D Stacked Bar | 2DHalf Bar | 3D Step | 3D Unstacked Area | 3D Ribbon | 3D Area | 3D Column | 3D Pie | 3D Contour | 3D Surface | 3D ShadedSurface
Rotate	Graph_Type	R3D bar | R2DHalf Bar | R2D Bar | Rotated Area | Rotated Line

Table B.14: Graph objects (fixed) (continued)

Object|Property	Argument	Syntax
Combo	Graph_Type	Area_bar | Hilo_bar | Line_bar | Multiple 3D columns | Multiple 3D Pies | Multiple Bar | Multiple Columns | Multiple Pies
Text	Graph_Type	Text
Legend Position	Legend_Position	None | Bottom | Right
3D View	3D_View	*Rotation, Elevation, Perspective, Depth, Height*
3D Options	3D_Options	*ShowLeft, ShowBack, ShowBase, ThickWalls*
Box Type	Box_Type	No box | Single outline | Rounded corners | Double outline | Thick outline 1 | Thick rounded corners | Three dimensional | Thick outline 2 | Bevel out | Shadowed | Thick outline 3 | Bevel in
Fill Color	Fill_Color	see page 402
Fill Style	Fill_Style	see page 404
Border Color	Border_Color	see page 402
Bkg Color	Bkg_Color	see page 402
Graph Button	Graph_Button	*GotoSlide?, SlideName, Effect, Duration,* Slow | Med | Fast, *Overlay?, RunMacro?, MacroText*
Name *	Name	
Graph Subtitle		
Subtitle Color	Subtitle_Color	see page 402
Subtitle Bkg Color	Subtitle_Bkg_Color	see page 402
Subtitle Font	Subtitle_Font	see page 404
Subtitle Style	Subtitle_Style	None | Solid | Wash | Bitmap, *Type, Shadow*
Solid	Subtitle_Style	Solid
Wash	Subtitle_Style	Wash,Left to right
	Subtitle_Style	Wash,Right to left
	Subtitle_Style	Wash,Center to left and right
	Subtitle_Style	Wash,Top to bottom
	Subtitle_Style	Wash,Bottom to top
	Subtitle_Style	Wash,Center to top and bottom
Bitmap	Subtitle_Style	Bitmap,Crop to fit | Shrink to fit,*BitmapName*
Graph Title		
Text Color	Text_Color	see page 402
Text Bkg Color	Text_Bkg_Color	see page 402
Text Font	Text_Font	see page 404
Text Style	Text_Style	see page 405
Graph Title Box		
Box Type	Box_Type	No box | Single outline | Rounded corners | Double outline | Thick outline 1 | Thick rounded corners | Three dimensional | Thick outline 2 | Bevel out | Shadowed | Thick outline 3 | Bevel in
Fill Color	Fill_Color	see page 402
Bkg Color	Bkg_Color	see page 402
Fill Style	Fill_Style	see page 404

Table B.14: Graph objects (fixed) (continued)

Object|Property	Argument	Syntax
Border Color	Border_Color	see page 402
Dimension	Dimension	see page 403
Name	Name	
Graph Window		
Aspect Ratio	Aspect_Ratio	Floating Graph | Screen Slide | 35mm Slide | Printer Preview | Full Extent
Grid	Grid	*GridSize, DisplayGrid, SnapToGrid*
Legend Box		
Legend Position	Legend_Position	None | Bottom | Right
Text Color	Text_Color	see page 402
Text Bkg Color	Text_Bkg_Color	see page 402
Text Font	Text_Font	see page 404
Text Style	Text_Style	see page 405
Box Type	Box_Type	No box | Single outline | Rounded corners | Double outline | Thick outline 1 | Thick rounded corners | Three dimensional | Thick outline 2 | Bevel out | Shadowed | Thick outline 3 | Bevel in
Fill Color	Fill_Color	see page 402
Bkg Color	Bkg_Color	see page 402
Fill Style	Fill_Style	see page 404
Border Color	Border_Color	see page 402
Pie Graph		
Explode Slice	Explode_Slice	*Distance, Explode*
Label Options	Label_Options	*Series*, Currency | Value | Percent | None, *ShowTick*
Text Color	Text_Color	see page 402
Text Bkg Color	Text_Bkg_Color	see page 402
Text Font	Text_Font	see page 404
Text Style	Text_Style	see page 405
Fill Color	Fill_Color	see page 402
Bkg Color	Bkg_Color	see page 402
Fill Style	Fill_Style	see page 404
Border Color	Border_Color	see page 402
Border Style	Border_Style	S0W1 | S0W2 | S0W3 | S0W4 | S1W1 | S2W1 | S3W1 | S4W1 | S5W1
Series Label		
Label Alignment	Label_Alignment	Above | Top | Center | Below for bar graphs Above | Center | Below | Left | Right for area and line graphs
Format	Format	*Format, Precision* | *Type*
Text Color	Text_Color	see page 402
Text Bkg Color	Text_Bkg_Color	see page 402
Text Font	Text_Font	see page 404
Text Style	Text_Style	see page 405

Table B.14: Graph objects (fixed) (continued)

Object|Property	Argument	Syntax
X-Axis		
Scale	Scale	Normal | Log, *Automatic, High, Low, Increment, #Minors, ShowUnits* **Note:** This property only appears when inspecting an XY graph
X-Axis Series	X-Axis_Series	
Tick Options	Tick_Options	None | Below | Above | Across, *DisplayLabels, NumRows, NoOverlap, #OfLabelsToSkip, Limit*
Numeric Format	Numeric_Format	*Format, Precision* | *Type* **Note:** This property only appears when inspecting an XY graph
Text Color	Text_Color	see page 402
Text Bkg Color	Text_Bkg_Color	see page 402
Text Font	Text_Font	see page 404
Text Style	Text_Style	see page 405
Major Grid Style	Major_Grid_Style	S0W1 | S0W2 | S0W3 | S0W4 | S1W1 | S2W1 | S3W1 | S4W1 | S5W1,*Red,Green,Blue*
Minor Grid Style	Minor_Grid_Style	S0W1 | S0W2 | S0W3 | S0W4 | S1W1 | S2W1 | S3W1 | S4W1 | S5W1,*Red,Green,Blue* **Note:** This property only appears when inspecting an XY graph
Y-Axis		
Scale	Scale	Normal | Log, *Automatic, High, Low, Increment, #Minors, ShowUnits* **Note:** An additional setting, *ZeroLine*, appears if the graph type is Variance.
Tick Options	Tick_Options	None | Left | Right | Across, *DisplayLabels, LengthLimit, Limit*
Numeric Format	Numeric_Format	*Format, Precision* | *Type*
Text Color	Text_Color	see page 402
Text Bkg Color	Text_Bkg_Color	see page 402
Text Font	Text_Font	see page 404
Text Style	Text_Style	see page 405
Major Grid Style	Major_Grid_Style	S0W1 | S0W2 | S0W3 | S0W4 | S1W1 | S2W1 | S3W1 | S4W1 | S5W1,*Red,Green,Blue*
Minor Grid Style	Minor_Grid_Style	S0W1 | S0W2 | S0W3 | S0W4 | S1W1 | S2W1 | S3W1 | S4W1 | S5W1,*Red,Green,Blue*

Notebook objects

You can use the following syntax to identify notebooks (not the objects in them):

[Notebook].Property

Notebook is the name of the notebook. *Property* is the property setting to read or change.

You can use block coordinates or names to set or study block property settings. For example, the following formula returns the numeric format of the block A1..A26 on page B of the active notebook:

@PROPERTY("B:A1..A26.Numeric_Format")

You can identify floating objects (graphs, macro buttons, and so on) in the active notebook by using the name of the object (found in its Object Name property). For example, if the active page contains a macro button with its Object Name property set to Button1, the following formula returns that macro button's Box Type:

@PROPERTY("Button1.Box_Type")

You can also use *PageName:ObjectName* to identify floating objects. For example, if the object in the previous example was on page AB, the following formula returns the macro button's Box Type:

@PROPERTY("AB:Button1.Box_Type")

The following table lists each notebook object and its properties:

Table B.15: Notebook objects

Object|Property	Argument	Syntax
Bitmap		
Border Color	Border_Color	see page 402
Box Type	Box_Type	None | Thin | Medium | Thick, *DropShadow*
Box Type	Box_Type.Frame_Line_Style	None | Thin | Medium | Thick
Drop Shadow	Box_Type.Drop_Shadow	
Object Name	Object_Name	
Block		
Numeric Format	Numeric_Format	*Format, Precision | Type*
Font	Font	see page 404
Shading	Shading	*Color1, Color2, Blend*
Color Blend 1	Shading.Color_1	0-15

Table B.15: Notebook objects (continued)

Object|Property	Argument	Syntax
Color Blend 2	Shading.Color_2	0-15
Select Color Blend	Shading.Blend	Blend1 | Blend2 | Blend3 | Blend4 | Blend5 | Blend6 | Blend7
Alignment	Alignment	General | Left | Right | Center
Line Drawing	Line_Drawing	*Left, Top, Right, Bottom, Vert, Horiz* (Each setting above can take NoChange | Clear | Thin | Thick | Double)
Protection	Protection	Protect | Unprotect
Text Color	Text_Color	0-15
Data Entry Input	Data_Entry_Input	General | Labels Only | Dates Only
Row Height	Row_Height	*Operation, Size*
Set Height	Row_Height	Set Height, *NewSize*
Reset Height	Row_Height	Reset Height
Column Width	Column_Width	*Operation, WidthInTwips, ColSpacing*
Set Width	Column_Width	Set Width, *NewWidthInTwips*
Reset Width	Column_Width	Reset Width
Auto Width	Column_Width	Auto Width,, *ExtraSpace*
Reveal/Hide	Reveal/Hide	Row | Column, Reveal | Hide
Style *	Style	The named style of the active block
Selection * (R)	Selection	The coordinates of the selected block
Value *	Value	The contents of the cell (as they appear on the input line)
Number Value *	Number_Value	The value in the cell
Button		
Macro	Macro	
Label Text	Label_Text	
Border Color	Border_Color	see page 402
Box Type	Box_Type	None | Thin | Medium | Thick, *DropShadow*
Box Type	Box_Type.Frame_Line_Style	None | Thin | Medium | Thick
Drop Shadow	Box_Type.Drop_Shadow	
Object Name	Object_Name	
Graph		
Source Graph	Source_Graph	
Border Color	Border_Color	see page 402
Box Type	Box_Type	None | Thin | Medium | Thick, *DropShadow*
Box Type	Box_Type.Frame_Line_Style	None | Thin | Medium | Thick
Drop Shadow	Box_Type.Drop_Shadow	
Object Name	Object_Name	
Notebook		
Recalc Settings	Recalc_Settings	Automatic | Manual | Background, Natural | Column-wise | Row-wise, *Iterations*
Zoom Factor	Zoom_Factor	25-200

Table B.15: Notebook objects (continued)

Object|Property	Argument	Syntax
Palette	Palette	*Color1,Color2,...,Color16*
Color 1	Palette.Color_1	see page 402
Color 2	Palette.Color_2	see page 402
Color 3	Palette.Color_3	see page 402
Color 4	Palette.Color_4	see page 402
Color 5	Palette.Color_5	see page 402
Color 6	Palette.Color_6	see page 402
Color 7	Palette.Color_7	see page 402
Color 8	Palette.Color_8	see page 402
Color 9	Palette.Color_9	see page 402
Color 10	Palette.Color_10	see page 402
Color 11	Palette.Color_11	see page 402
Color 12	Palette.Color_12	see page 402
Color 13	Palette.Color_13	see page 402
Color 14	Palette.Color_14	see page 402
Color 15	Palette.Color_15	see page 402
Color 16	Palette.Color_16	see page 402
Display	Display	*VertScroll, HorScroll, Tabs*
Vertical Scroll Bar	Display.Show_VerticalScroller	
Horizontal Scroll Bar	Display.Show_HorizontalScroller	
Page Tabs	Display.Show_Tabs	
Macro Library	Macro_Library	
Group Mode *	Group_Mode	On enables Group Mode; Off disables group mode
Password *	Password	the password of the active notebook
OLE		
Object settings	none	none
Link settings	Link_Settings	*UpdateType*
Border Color	Border_Color	see page 402
Box Type	Box_Type	None | Thin | Medium | Thick, *DropShadow*
Box Type	Box_Type.Frame_Line_Style	None | Thin | Medium | Thick
Drop Shadow	Box_Type.Drop_Shadow	
Object Name	Object_Name	

Graphs page icons

When the Graphs page is active you can change the properties of icons on the Graphs page instead of the objects they represent. For example, the following macro sets the Show Pointer property of a slide show named Presentation2 to Yes:

```
{SETOBJECTPROPERTY "Presentation2.Show_Pointer","Yes"}
```

The following table lists the properties of these icons.

Table B.16
Graphs page icons

Object|Property	Argument	Syntax
Graph Icon	Name	
Slideshow Icon		
Name	Name	
Default Effect	Default_Effect	*Effect*, *DisplayTime*, Slow | Med | Fast, *Overlay?*
Show Pointer	Show_Pointer	
Dialog Icon		
Dimension	Dimension	see page 403
Title	Title	
Position Adjust	Position_Adjust	*Depend*, *LeftRel*, *TopRel*, *RightRel*, *BottomRel*, *CenterHor*, *CenterVer*
Grid Options	Grid_Options	*Gridsize*, *GridShown*, *GridEnabled*
Name	Name	
Disabled	Disabled	
Enabled *	Enabled	
Value *	Value	

A P P E N D I X

C

ANSI codes

The American National Standards Institute (ANSI) character set is the accepted standard for translating alphabetic and numeric characters, symbols, and control instructions into 8-bit binary code. Table C.1 shows all ANSI characters with their decimal and hexadecimal equivalents.

Note In addition to the characters on your keyboard, you can enter any ANSI character in a notebook or graph. This includes international characters such as ñ and á, mathematical symbols such as ±, and even special symbols such as ¶ and þ.

To enter a character into a notebook using its ANSI code,

You can also activate the input line by clicking it or pressing F2.

1. Locate the character in Table C.1 and note the number next to it in the Dec column. This is the character's decimal number.
2. Type a label prefix character (', ", or ^) to activate the input line.
3. Hold down the *Alt* key.
4. Using the numeric keypad, type zero followed by the character's decimal number. Don't press *Enter*.
5. Release the *Alt* key. The character appears on the input line.
6. Press *Enter* to place the character in the cell.

Table C.1
ANSI character set

Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char
0	0	▮	32	20	*(space)*	64	40	@	96	60	`
1	1	▮	33	21	!	65	41	A	97	61	a
2	2	▮	34	22	"	66	42	B	98	62	b
3	3	▮	35	23	#	67	43	C	99	63	c
4	4	▮	36	24	$	68	44	D	100	64	d
5	5	▮	37	25	%	69	45	E	101	65	e
6	6	▮	38	26	&	70	46	F	102	66	f
7	7	▮	39	27	'	71	47	G	103	67	g
8	8	▮	40	28	(	72	48	H	104	68	h
9	9	▮	41	29	)	73	49	I	105	69	i
10	A	▮	42	2A	*	74	4A	J	106	6A	j
11	B	▮	43	2B	+	75	4B	K	107	6B	k
12	C	▮	44	2C	,	76	4C	L	108	6C	l
13	D	▮	45	2D	-	77	4D	M	109	6D	m
14	E	▮	46	2E	.	78	4E	N	110	6E	n
15	F	▮	47	2F	/	79	4F	O	111	6F	o
16	10	▮	48	30	0	80	50	P	112	70	p
17	11	▮	49	31	1	81	51	Q	113	71	q
18	12	▮	50	32	2	82	52	R	114	72	r
19	13	▮	51	33	3	83	53	S	115	73	s
20	14	▮	52	34	4	84	54	T	116	74	t
21	15	▮	53	35	5	85	55	U	117	75	u
22	16	▮	54	36	6	86	56	V	118	76	v
23	17	▮	55	37	7	87	57	W	119	77	w
24	18	▮	56	38	8	88	58	X	120	78	x
25	19	▮	57	39	9	89	59	Y	121	79	y
26	1A	▮	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	▮

▮ = character not defined in Windows

Table C.1: ANSI character set (continued)

Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char
128	80	▮	160	A0	*(space)*	192	C0	À	224	E0	à
129	81	▮	161	A1	¡	193	C1	Á	225	E1	á
130	82	‚	162	A2	¢	194	C2	Â	226	E2	â
131	83	ƒ	163	A3	£	195	C3	Ã	227	E3	ã
132	84	„	164	A4	¤	196	C4	Ä	228	E4	ä
133	85	…	165	A5	¥	197	C5	Å	229	E5	å
134	86	†	166	A6	¦	198	C6	Æ	230	E6	æ
135	87	‡	167	A7	§	199	C7	Ç	231	E7	ç
136	88	ˆ	168	A8	¨	200	C8	È	232	E8	è
137	89	‰	169	A9	©	201	C9	É	233	E9	é
138	8A	Š	170	AA	ª	202	CA	Ê	234	EA	ê
139	8B	‹	171	AB	«	203	CB	Ë	235	EB	ë
140	8C	Œ	172	AC	¬	204	CC	Ì	236	EC	ì
141	8D	▮	173	AD	–	205	CD	Í	237	ED	í
142	8E	▮	174	AE	®	206	CE	Î	238	EE	î
143	8F	▮	175	AF	—	207	CF	Ï	239	EF	ï
144	90	▮	176	B0	°	208	D0	Ð	240	F0	ð
145	91	‘	177	B1	±	209	D1	Ñ	241	F1	ñ
146	92	’	178	B2	²	210	D2	Ò	242	F2	ò
147	93	“	179	B3	³	211	D3	Ó	243	F3	ó
148	94	”	180	B4	´	212	D4	Ô	244	F4	ô
149	95	•	181	B5	µ	213	D5	Õ	245	F5	õ
150	96	–	182	B6	¶	214	D6	Ö	246	F6	ö
151	97	—	183	B7	·	215	D7	×	247	F7	÷
152	98	˜	184	B8	¸	216	D8	Ø	248	F8	ø
153	99	™	185	B9	¹	217	D9	Ù	249	F9	ù
154	9A	š	186	BA	º	218	DA	Ú	250	FA	ú
155	9B	›	187	BB	»	219	DB	Û	251	FB	û
156	9C	œ	188	BC	¼	220	DC	Ü	252	FC	ü
157	9D	▮	189	BD	½	221	DD	Ý	253	FD	ý
158	9E	▮	190	BE	¾	222	DE	Þ	254	FE	þ
159	9F	Ÿ	191	BF	¿	223	DF	ß	255	FF	ÿ

▮ = character not defined in Windows

INDEX

1-2-3
  macros *116*
(:) colon
  relative references *137*
"" (empty strings) *70, 89, 90*
{ } macro command *156*
& (ampersand)
  command definitions *375*
' (apostrophe) label prefix
  macro entry *117*
@ (at-sign) *3*
\ (backslash)
  naming macros *113*
: (colon)
  command definitions *377*
  macro suffix *122*
, (comma)
  as argument separator *24, 120*
! (exclamation point)
  in macros *130*
! (exclamation point) in formulas *90*
@@ function *21*
– (hyphen)
  command definition *380*
? (question mark)
  in macros *130*
  macro command *157*
" (quote mark)
  macro arguments *121*
; (semicolon)
  in comments *6, 156*
  macro command *156*
~ (tilde)
  command *155, 178*
@Functions button *4*
/x macro commands *123, 155*

A

@ABS function *22*
ABS macro command *157*
absolute cell address *157*
  in macros *110*
absolute values *22*
accelerated depreciation *39, 96*
accelerators *See* shortcut keys
access modes (files) *229*
@ACOS function *22*
action boxes *361*
actions, performing *315, 320, 339, 352-364*
ACTIVATE macro command *158*
add-in @functions *20*
adding
  macro commands *159*
  menus *159*
addition *8*
ADDMENU macro command *159, 373*
ADDMENUITEM macro command *159, 373, 379*
addresses
  absolute
    in macros *110*
  cell
    combining with ampersands *68*
alarms *352*
alert boxes *222*
Align command, Dialog menu *319, 326*
Align.*Option* command equivalent *161*
aligning objects *161*
alphabetical order in strings *8*
ALT macro command *161*
amortized payments *79*
ampersand
  combining strings or cell addresses *8, 68*
  command definitions *375*
angles
  arc cosine *22*

arc sine *23*
arc tangent *23*
cosine *31*
degrees, converting to radians *85*
radians, converting to degrees *41*
sine *91*
tangent *97*
trigonometric @functions and *10, 12*
annual interest rate *63, 80*
annuity, ordinary
calculating *33, 80, 98*
defined *15*
financial @functions *14*
future value *52*
present value *83*
annuity due
calculating *75, 78*
defined *15*
financial @functions *14*
future value *53*
present value *84*
ANSI codes *431*
converting strings to *30*
converting to characters *28*
returning *30*
apostrophe label prefix (')
macro entry *117*
appending files *230*
application building
changing properties *296*
defined *283-291*
dialog boxes *186, 291-293*
macros *296*
menus *294*
SpeedBars *293*
Application properties *162*
Application.*Property* command equivalent *162*
applications, other
closing conversations with *272*
communicating with *131, 213*
getting information from *253*
sending data to *240*
running *189*
macros *189*
arc cosine *22*
arc sine *23*
arc tangent *23*
arguments *7*
@function *4*
blank cells *10*
database @functions and *12*
financial @functions and *14*
@function *4, 5, 6*
logical @functions and *13*
macros *121*
passing to subroutines *126, 180, 269*
mathematical @functions and *10, 12*
separators *24, 49*
statistical @functions and *10*
arithmetic operators *8*
precedence *7*
arranging
window icons *275*
arrow
creating *178*
ASCII
accessing files *211*
@ASIN function *23*
assets
depreciation
accelerated *39, 96*
straight-line allowance *92*
future value *86*
calculating *52, 53*
calculating time periods *33, 75, 98*
net present value *76*
present value
calculating *83, 84*
at-sign *3*
@ATAN2 function *23*
@ATAN function *23*
attached controls *335*
moving *336-338*
attributes
in arguments *25, 27*
autoload macro *115, 296*
averages *24*
fields *36*
@AVG function *24*

B

backslash (\)
naming macros *113*
BACKSPACE macro command *165*

BACKTAB macro command *166*
backward, sending controls *326*
balloon payments *79*
base 10 logarithms *69*
BEEP macro command *166*
beeps
  macro command *166*
BIGLEFT macro command *166*
BIGRIGHT macro command *167*
bitmap
  buttons
    creating *319, 331, 341*
bitmap buttons *See also* buttons
bitmaps
  macro buttons and *113*
blank
  cells
    in arguments *10*
    macros *111, 117, 125*
    testing for *66*
  lines
    inserting in macros *156*
BLANK macro command *167*
block names
  @functions and *5*
  macros *117-118*
block values
  as arguments *6*
  averaging *24*
  maximum *69*
  minimum *71*
BlockCopy command equivalent *167*
BlockDelete.*Option* command equivalent *168*
BlockFill.*Option* command equivalent *168*
BlockInsert.*Option* command equivalent *169*
BlockMove command equivalent *169*
BlockMovePages command equivalent *170*
BlockName.*Option* command equivalent *170*
BlockReformat command equivalent *171*
blocks
  columns, finding number of *30*
  converting to formulas to values *171*
  copying *167, 188*
  database @functions *13*
  deleting *168, 187, 188*
  dot (scalar) product *96*
  erasing *167*
  extracting to files *192*
  filling *267*
  filling with data *168*
  formatting *268*
  index *59*
  inserting *169, 188*
  macros *122, 136*
    storing *108*
  moving *169*
  naming *170*
  non-blank cells, finding *32*
  pages, finding number of *91*
  reformatting *171*
  rows, returning number of *89*
  searching horizontally *54*
  searching vertically *104*
  selecting *188, 259*
  transposing *171*
BlockTranspose command equivalent *171*
BlockValues command equivalent *171*
book value, calculating *39*
borrower rate vs. lender rate *64*
BRANCH macro command *127, 171*
branching macros *125, 127, 172, 185*
BREAK macro command *172*
BREAKOFF macro command *112, 172*
BREAKON macro command *173*
breakpoints *140*
  removing *144*
bringing controls forward *326*
BS macro command *165*
Budgeteer *284-291*
BUDGTRAK.WB1 *284*
Button tool, SpeedBar *113*
buttons *See also* specific type
  labeling *319, 331*
  properties *340*
  tools *304, 307*
  types *307*
bytes
  calculating in files *193*
  reading in files *248, 249*

C

Calc key (F9)
  macro equivalent *173*

CALC macro command *173*
Cancel button *303*
  creating *342*
CAPOFF macro command *173*
CAPON macro command *173*
carriage returns
  characters *248, 280*
    inserting in strings *281*
  macros *155, 178*
cascading
  windows *275*
cascading menus *See* submenus
case sensitivity
  @functions *5*
  macros *113, 120*
cash flow
  financial @functions *16*
  internal rate of return *62*
  multiple rates of return *63*
  net present value *76*
  simple transactions *62*
  table *62, 76, 77*
@CELL function *25*
cell macro commands *122, 127, 145, 149*
cell selector
  moving
    down *187*
    left *166, 215*
    right *167, 256, 270*
    up *272*
@CELLINDEX function *27*
@CELLPOINTER function *27*
cells
  addresses
    absolute *157*
    absolute vs. relative *110*
    ampersands and *8, 68*
  attributes *25*
    returning *25, 28*
  blank
    in arguments *10*
    macros *111, 117, 125*
    testing for *66*
  copying *176*
  entering
    macros as labels in *117*
    values *216, 244, 246*
      same repeated *245*
  erasing *167*
  linking to controls *339*
  non-blank, finding *32*
  selecting
    leftmost *166*
    right of current cell *167, 270*
  testing content *28, 65-67*
  tracing *143*
@CHAR function *28*
character strings *18*
  converting to numeric values *101*
  copying *280, 281*
  reading *212*
  relational operators and *8*
  string @functions and *18*
characters
  ANSI codes
    converting to *30*
    returning from *28*
  CR/LF *248, 280*
    inserting in strings *281*
  deleting
    macro command *165*
  extracting from strings *67, 70, 88*
  lowercase, changing to *69*
  proper case, changing to *82*
  repeating *87*
  replacing *87*
  uppercase, changing to *101*
check box *307, 333*
  bitmap *342*
  properties *340*
check boxes
  creating *341*
check marks by menu commands *379*
child control *335*
  moving *336-338*
choice lists
  @functions *4*
Choices key (F3)
  macro equivalent *224*
@CHOOSE function *29*
CHOOSE macro command *173*
circles, determining area *79*
@CLEAN function *30*
CLEAR macro command *173*

ClearContents command equivalent 174
client, DDE conversation 131
Clipboard
dialog window 304
labeling controls 331
moving controls 324
clock, displaying 351-352
CLOSE macro command 174
closing
files 191
windows 275
@CODE function 30
colon (:)
command definitions 377
macro suffix 122
relative references 137
color
dialog box
rectangle 346
setting 349
color controls 349
@COLS function 30
columns
database @functions 13
hiding 256
in block 30
inserting 169
revealing 256
totaling 268
width
setting 174
specifying 176
COLUMNWIDTH macro command 174
columnwise recalculation 251
combining
files 191
combo boxes 347
comma
as argument separator 24, 120
@COMMAND function 30
command definition 373
menu command 374
syntax 374
command equivalent macros 152
command equivalents 123, 129
macro libraries and 137
command equivalents macro commands 146
command settings, current value 34
commands
macro *See* macro commands
menu *See* menu commands
comments, hidden 6, 156
commission
calculating average 24
calculating lowest 71
compound conditions, operators and 9, 57
compound interest 33, 52, 75, 83, 98
condition argument (macros) 122
conditional breakpoints 140, 142
removing 144
standard vs. 140
conditional statements 13
Connect command, Dialog menu 339
CONTENTS macro command 176
control boxes
action 361
events 353-355
control panel area
disabling 236
enabling 236
controls *See also* objects, 301, 304
aligning 326
arranging 327, 328
attached 335
moving 336-338
bitmap buttons 341
check boxes 333, 340, 342
color controls 349
combo boxes 347
connecting 339
copying 324
creating
dialog box 307, 308
SpeedBar 319
customizing 311, 324-340
dimension, setting 325
disabling 334-335
edit fields 344-345
file controls 349
group boxes 308, 309, 346
grouping 309, 346
hiding 334-335
labeling 311, 319, 331, 342-344
linking to actions 339, 352-364

dialog box *315*
SpeedBar *320*
list boxes *347*
moving *324*
ordering *333*
settings *330, 331*
overlapping *326*
parent and child *335*
moving *336-338*
pick lists *348*
properties *311, 324, 334-335, 340*
push buttons *319, 340, 342*
radio buttons *309, 333, 340, 341, 342*
rectangles *308, 335, 346*
resizing *325*
scroll bars *350*
selecting *305, 324*
settings *329, 330, 331*
status line message, creating *324*
Tab key *333*
testing *304, 317, 323*
time control *351-352, 364*
types *307, 340*
Controls.*Option* command equivalent *177*
conversations
closing *272*
DDE *131*
initiating *213*
converting
text to values *237*
coordinates, block
macro commands *122*
copying
blocks *167*
contents and properties *238*
graphs *205*
to and from the Clipboard *188*
@COS function *31*
cosine *31*
@COUNT function *32*
counting
data *198*
CR macro command *178*
CREATEOBJECT macro command *178*
creating
files *192*
graphs *206*
workspaces *280*
@CTERM function *32*
Ctrl+Break key *112*
disabling/enabling *172*
CTRL macro command *179*
@CURVALUE function *34*
Cut, Copy, and Paste
controls *324, 331*
dialog window SpeedBar *304*
cutting
data *188*

D

D macro command *187*
data
accessing
in other files *136*
copying *176, 188*
counting *198*
deleting *174, 187, 188*
entry *202*
restricting *344-345*
importing *101*
inserting *188*
parsing *237*
sorting *267*
storing in cells *216, 244, 246*
same repeated *245*
tables *244*
data entry
restricting *255*
database @functions *12*
vs. statistical @functions *12*
databases
external *271*
querying *247*
date
current, entering *100*
current, returning *75*
day of month, returning *37*
formats *35*
standard *35*
storing in notebooks *17*
macro command *179*
month, returning *72*
prefix, entering *36*

serial values, returning *34, 35*
strings, converting to *94*
@SUM, with *95*
year, returning *105*
date @functions *17*
@DATE function *34*
DATE macro command *179*
@DATEVALUE function *35*
@DAVG function *36*
@DAY function *37*
@DCOUNT function *38*
@DDB function *39*
DDE
applications *See* applications, other
commands *189, 213, 240, 253, 272*
conversations *131, 214*
closing *272*
macro commands *123, 131, 145, 150*
DDE links
creating *238*
@DDELINK function *40*
Debug
key (Shift+F2) *139, 144, 269*
mode *139, 269*
window *140*
DEBUG indicator *139*
Debugger command *139, 144*
debugging *See* macros, debugging
decimal codes, ANSI characters *431*
decimal numbers
converting hexadecimal numbers to *54*
default settings
startup macros *115*
DEFINE macro command *126, 180*
definition *See* command definition
degrees *10*
converting to radians *85*
@DEGREES function *41*
Del key
macro equivalent *181*
DEL macro command *181*
DELETE macro command *181*
DELETEMENU macro command *181, 373, 379*
DELETEMENUITEM macro command *182, 373*
deleting
blocks *168*
data *188*
graphs *206*
menu commands *182*
menus *181, 379*
deleting data *174, 187*
depreciation
accelerated *39, 96*
financial @functions *16*
straight-line allowance *92*
designing applications *See* application building
Developer mode *298*
dialog boxes *222, 301*
clearing *172*
controls *See* controls
creating *See also* controls; tools, *291-293, 302-312, 366*
customizing *311, 324-340*
displaying *312, 329*
editing *308-323*
hot keys *See* shortcut keys
letter, underlined *332*
names *303, 306*
objects *See also* controls
creating *178*
identifying *408*
properties *263, 408*
selecting *260*
rectangles *308, 335, 346*
resizing *312*
saving *302, 307*
scroll bars *350*
shortcut keys *332*
status line message, creating *324*
Tab key *333*
testing *304, 317, 323*
text, adding *311, 331, 342-344*
titles *304, 307*
window, dialog *See* dialog window
dialog controls *See* controls
ordering *177*
dialog icon *307*
Dialog menu
Align *319, 326*
Connect *339*
Links *315, 320, 352*
Order *326, 330*
Save SpeedBar As *318*
dialog objects *See* objects; controls

dialog window *303*
  Clipboard *304*
  opening *306*
  resizing objects *255*
  SpeedBar *304, 307*
dialog windows
  ordering objects *233*
dialogs *See* dialog boxes
DialogView command equivalent *182*
DialogWindow.*Property* command equivalent *182*
Dimension options *325*
dimming menu commands *372, 378*
directory
  checking file *50*
  dialog box display *349*
DISPATCH macro command *185*
displaying
  graphs *208*
  hidden windows *277*
dividends *63*
divider lines *372*
  inserting *380*
dividing
  windows into panes *277*
division *8*
DLL
  @functions *20*
  macros *216*
@DMAX function *42*
@DMIN function *43*
DODIALOG macro command *186, 239, 292*
  displaying dialog boxes *312, 329*
DOMACRO link command *376*
DOS prompt, running macros from *112*
dot product *96*
double-declining depreciation allowance *39*
DOWN macro command *187*
drawings *See* objects
drop-down list boxes *See* combo boxes
@DSTD function *44*
@DSTDS function *45*
@DSUM function *46*
@DVAR function *47*
@DVARS function *48*
Dynamic Data Exchange *See* DDE

E

edit fields *308, 344-345*
Edit key (F2)
  macro equivalent *187*
EDIT macro command *187*
EditClear command equivalent *187*
EditCopy command equivalent *188*
EditCut command equivalent *188*
EditGoto command equivalent *188*
editing
  graphs *206, 207*
  macros *143*
EditPaste command equivalent *188*
ellipses
  creating *178*
  menu titles and *382*
empty strings *70, 89, 90*
End key
  macro equivalent *188*
END macro command *188*
Enter key
  macro equivalent *178*
entering macros as labels *117-119*
@ERR function *49*
ERR values *9*
  finding *65*
  returning *49*
errors
  during macro execution *227*
  handling *227, 255*
ESC and ESCAPE macro commands *115, 188*
Esc key
  custom menus *219*
  macro equivalent *188*
events *353-355*
  timing *351-352, 364*
@EXACT function *49*
exclamation point (!)
  in macros *130*
exclamation point in formulas (!) *90*
EXEC macro command *189*
EXECUTE link command *361, 376*
EXECUTE macro command *189*
exiting
  debugger *144*
  files *191*
@EXP function *50*

expanded memory, returning available *70*
exponentiation *8*
ExportGraphic command equivalent *190*
exporting
  graphics *190*
extracting
  blocks to files *192*
  files *192*

F

@FALSE function *50*
fields
  averaging *36*
  counting entries in *38*
  database @functions *12*
  maximum, finding *42*
  minimum, finding *43*
  standard deviation *44, 45*
  totaling *46*
  variance *47, 48*
file controls *349*
file macro commands *122, 128, 145, 150*
FileClose... command equivalents *191*
FileCombine command equivalent *191*
@FILEEXISTS function *50*
FileExit command equivalent *191*
FileExtract command equivalent *192*
FileImport command equivalent *192*
FileNew command equivalent *192*
FileOpen command equivalent *192*
FileRetrieve command equivalent *193*
files
  accessing *229*
  closing *174, 191*
  combining *191*
  copying character strings to *280, 281*
  creating *192*
  dialog box display *349*
  directory
    checking *50*
    storing *200*
  exiting *191*
  extracting *192*
  importing text *192*
  opening *192*
  pointer *202, 264*
  retrieving *193*
  saving *193*
  size in bytes
    calculating *193*
    reading *248, 249*
FileSave... command equivalents *193*
FILESIZE macro command *193*
filling blocks *168*
financial @functions *14*
  annuity *14*
  arguments *14*
  cash flow *16*
  depreciation *16*
@FIND function *51*
FLOATCREATE macro command *194*
floating objects
  creating *194*
  moving *195*
  ordering *196*
  resizing *196*
  selecting *259*
FLOATMOVE macro command *195*
FloatOrder.*Option* command equivalent *196*
FLOATSIZE macro command *196*
flow charts *118*
FOR macro command *197*
FORBREAK macro command *198*
formatting
  data *224*
formulas
  comments in *6*
  converting to values *171*
  logical
    in macros *122, 142*
  operators *8*
    precedence *7*
  recalculating
    during macros *173, 250*
  statistical @functions *11*
  text
    in macros *117, 121, 138*
forward, bringing controls *326*
Frequency.*Option* command equivalent *198*
@functions *3-5*
  add-in *20*
  arguments *5, 6*
  case sensitivity *5*
  choosing *4*

comments, adding *6*
database *12*
date and time *17*
entering *4-9*
financial *14*
help *4*
logical *13*
macros and *120*
determining cell contents *28*
numeric format display *94*
returning command settings *34*
mathematical *10*
maximum length *5, 6*
miscellaneous *19*
nesting *5*
parentheses *5*
positive/negative signs and *16*
statement *4*
statistical *11*
string *18*
syntax *4-9*
types *3*
Functions key (Alt+F3) *4*
macro equivalent *199*
FUNCTIONS macro command *199*
future value *86*
calculating *52, 53*
calculating time periods *33, 75, 98*
principal *78*
@FV function *52*
@FVAL function *53*

G

GET macro command *199*
GETDIRECTORYCONTENTS macro command *200*
GETLABEL macro command *200*
GETNUMBER macro command *201*
GETOBJECTPROPERTY macro command *202*
GETPOS macro command *202*
GETPROPERTY macro command *203*
GETWINDOWLIST macro command *204*
goal-seeking problems *231, 266*
GOTO macro command *204*
graph buttons
running macros *115*
Graph key (F11)
macro equivalent *204*
GRAPH macro command *204*
graph windows
properties *209*
GRAPHCHAR macro command *204*
GraphCopy command equivalent *205*
GraphDelete command equivalent *206*
GraphEdit command equivalent *206*
graphics
exporting *190*
importing *212*
graphics characters, ANSI *28*
GraphNew command equivalent *206*
GRAPHPAGEGOTO macro command *207*
graphs
copying *205*
creating *206*
current
displaying *204*
deleting *206*
editing *206, 207*
floating
creating *194*
moving *195*
resizing *196*
selecting *259*
grouping objects *210*
objects
properties *262, 263*
objects (drawn)
identifying *417*
properties *417*
objects (fixed)
identifying *420*
properties *420*
ordering objects *233*
printing *241*
properties *207*
series *260*
type *207*
ungrouping objects *272*
viewing *208*
Graphs page
dialog box, creating *307*
moving icons *223*
SpeedBar *307, 318*

GraphSettings.Titles command equivalent 207
GraphSettings.Type command equivalent 207
GraphView command equivalent 208
GraphWindow.*Property* command equivalent 209
grid, aligning controls 326
group boxes 308, 309, 346
Group.*Option* command equivalent 210
grouping
- graph objects 210
- pages 210

grouping controls 335
GroupObjects command equivalent 210

H

help
- @functions 4

HELP macro command 210
hexadecimal numbers 78
- converting to decimal numbers 54

@HEXTONUM function 54
hidden comments
- in @functions 6
- in macros 156

hiding
- windows 276

hints
- menu commands 377
- status line 324, 372

HLINE macro command 210
@HLOOKUP function 54
HOME macro command 211
horizontal scroll bars 350
hot keys *See* shortcut keys
@HOUR function 56
HPAGE macro command 211
hyphens
- command definition 380

I

@IF function 57
IF macro command 211
IFKEY macro command 212
ImportGraphic command equivalent 212
importing
- data 237
- graphics 212
- text files 192

index
- blocks 58
- string functions 51, 70, 87
- tables 27, 54, 58, 103
- values 27, 55

@INDEX function 58
INDICATE macro command 212
indirect references 185
INITIATE macro command 213
input boxes *See* edit fields
input line
- erasing 173
- updating 201, 236

INSERT macro command 214
InsertBreak command equivalent 214
inserting
- blocks 169
- columns 169
- from the clipboard 188
- OLE objects 214
- page breaks 214
- pages 169
- rows 169

InsertObject command equivalent 214
INSON and INSOFF macro commands 214
Inspector, Object
- controls 340
- SpeedBars 321, 322

@INT function 59
integers
- returning 59
- rounding to 89

interactive macro commands 145, 148
interactive macros 124
- beeps and 166
- commands 122, 124
- controlling 172
- pausing macro for user input 157, 199
- prompting
  - for label 200
  - for numeric value 201

interest
- vs. principal portion of payment 60, 81

interest rates
- annual 63, 80

calculating *61, 86*
compounding periods *33, 75, 98*
internal rate of return *62*
Invert.*Option* command equivalent *215*
inverting
matrixes *215*
@IPAYMT *60*
@IRATE *61*
@IRR function *62*
@ISERR function *65*
@ISNA function *66*
@ISNUMBER function *66*
@ISSTRING function *67*

K

keyboard macro commands *122, 123, 145, 146*
keys
in dialog boxes *See* shortcut keys
keystroke macros *109, 123*
keystrokes
running macros with *113*
testing for *205, 212, 217*

L

label entry vs. value entry
in macros *122*
label-prefix characters *117*
Label tool, dialog window SpeedBar *311*
labeling
buttons *319, 331*
dialog box controls *311, 331, 342-344*
labels
characters
extracting *70*
left *67*
right *88*
finding number *68*
replacing *87*
comparing *49*
converting numeric values to *94*
creating *197*
entering *245, 246*
macros as *117-119*
prompting for *200*
statistical @ functions and *11*
string @functions and *18*
testing for *28, 67, 73*
translating block references *21*
@LEFT function *67*
LEFT macro command *215*
lender rate vs. borrower rate *64*
@LENGTH function *68*
LET macro command *216*
letter, underlined
dialog boxes *332*
menus *372*
linear programming *215, 223, 231*
linefeed characters *248, 280*
inserting in strings *281*
lines
blank
inserting in macros *156*
calculating length *249*
creating *178*
link commands *See also* linking, *291, 315, 352-364, 372*
application building *293*
command definitions *375*
control boxes *See* control boxes
dialog box controls *315, 339*
performing actions with *352-364*
SpeedBar controls *320, 339*
testing *304, 317, 323*
linking
controls *339, 352-364*
DDE objects *238*
dialog box controls *315*
events, trapping *353-355*
macros and *110, 136*
menus *375*
notebooks *217*
OLE objects *237*
SpeedBar controls *320*
linking objects *229*
Links command, Dialog menu *315, 320, 352*
Links.*Option* command equivalent *217*
list boxes *347*
drop-down *See* combo boxes
properties *347*
@LN function *68*
loading
files *192*
workspaces *280*

LOANPMT.WB1 *329*
location argument (macros) *121*
locked titles
  creating and clearing *279*
@LOG function *69*
logarithms
  base 10, returning *69*
  natural, returning *68*
logical @functions *13, 65-67*
  false (0), returning *50*
  true (1), returning *101*
logical expressions *9, 13*
  compound conditions *57*
  debugging and *142*
  evaluating *57, 211*
logical macro recording mode *109*
logical operators *9*
LOOK macro command *217*
lookup tables *73*
loops *197*
  debugging macros *141, 142*
@LOWER function *69*
lowercase characters *69*
  changing to proper case *82*
  changing to uppercase *101*

M

macro buttons
  creating *194, 340*
  moving *195*
  resizing *196*
  running macros from *113*
  selecting *259*
Macro command
  Tools menu *108*
macro commands *See also* specific commands, *108, 122, 119-136*
  adding *159*
  cell
    definition *145*
    list *149*
  command equivalents
    definition *146*
    list *152*
  DDE
    definition *145*
    list *150*
  dialog box display *186, 292*
  entering *120*
  file
    definition *145*
    list *150*
  inserting *123*
  interactive
    definition *145*
    list *148*
  keyboard
    definition *145*
    list *146*
  menu building *294*
  miscellaneous
    definition *145*
    list *151*
  object
    definition *145*
    list *151*
  program flow
    definition *145*
    list *148*
  property changing *296*
  recording *108-110*
  screen
    list *147*
  syntax *120*
  types *1-154*
  UI building
    definition *145*
    list *150*
  using multiple *119*
  /x *123, 155*
MACRO indicator *111*
macro libraries *110-111*
  running macros from *136*
macros *107, 118*
  1-2-3 *116*
  @functions and
    determining cell contents *28*
    numeric format display *94*
    returning command settings *34*
  addressing *122, 136*
    absolute vs. relative *110*
  arguments *121-122*
  autoload *115, 296*
  branching *125, 172, 185*

building applications *296*
buttons *See* macro buttons
carriage returns *155, 178*
command equivalents
  keystroke vs. *109*
commands *See* macro commands, *119-138*
commenting *119*
compatibility *155*
debugging *139-144, 269*
  breakpoints and *140-143*
  Debug mode *139*
  disabling *144*
  enabling *139*
  macro loops *141, 142*
  menu *140*
  trace cells and *143*
  window *139, 140*
dialog boxes, displaying *312, 329*
displaying menus *221*
editing *143*
entering as labels *117-119*
execution *See* macros, running
@functions *120*
graph buttons *115*
hiding commands *156*
interactive *See* interactive macros
keystroke *123*
  command equivalents vs. *109*
libraries *See* macro libraries
linking *136*
logical formulas *122*
loops *141, 142, 197*
Macros key *122*
menus
  displaying *219*
naming *113, 117-118*
pausing *140, 157, 199, 239, 273*
  for label *200*
  for numeric value *201*
  message *222*
Quattro Pro for DOS *116, 146*
recording *108-110*
running *171*
  at DOS prompt *112*
  changing macro while *138*
  errors *227*
  from buttons *113*
  from objects *115*
  from Windows *112*
  in Debug mode *140*
  in different notebooks *136*
  in other applications *189*
  type-ahead buffer *217*
  without interruption *172*
screen updating *236, 279*
  disabling *236, 278*
self-modifying *138*
speed, improving *236, 278*
stepping through *140*
stopping *112, 117, 144, 198, 247*
storing *108, 110-111*
subroutines *125*
typing *117-136*
Macros button *146*
Macros key (Shift+F3) *122, 146, 155*
MACROS macro command *218*
MARK macro command *219*
mathematical @functions *10*
mathematical constant *50, 68*
matrixes
  inverting *215*
  multiplying *223*
@MAX function *69*
maximizing
  windows *276*
@MEMAVAIL function *70*
@MEMEMSAVAIL function *70*
memory
  conventional, available *70*
  expanded, available *70*
menu bar *372*
  adding menus *373-374*
  displaying *262*
  replacing *374*
menu blocks *373*
  building *219*
menu commands *371, 372*
  activating *375*
  adding and deleting *379*
  check marks *379*
  command definitions *373*
  creating *374*
  deleting *182*
  dimming *372, 378, 382*

disabling *378*
graying *See* dimming
grouping *382*
inserting *374*
macros
keystroke *109*
menu-equivalent commands *116, 123, 129*
menu path *372*
menu titles
ellipses and *382*
MENUBRANCH macro command *219*
MENUCALL macro command *221*
menus *371, 372*
adding *159*
to menu bar *373-374*
blocks *See* menu blocks, *219, 373*
building
in macros *219, 220*
command settings, value *34*
commands *See* menu commands
creating *294, 1-382*
debug window *140*
deleting *181, 379*
dimmed commands *372*
displaying *262*
macro execution *219, 221*
divider lines *372*
editing *373-382*
graphs functioning as *205*
macros
keystroke *109*
properties *263*
shortcut keys *372*
underlined letter *372*
updating *201, 236*
using *374*
in macros *219*
message, status line *See* hints
message box *204, 222*
MESSAGE macro command *205, 222*
@MID function *70*
military hours *56*
@MIN function *71*
minimizing
windows *276*
@MINUTE function *71*
miscellaneous @functions *19*
miscellaneous macro commands *123, 145, 151*
@MOD function *72*
mode indicator *212*
modes
Developer *298*
macro recording *109*
modifying files *230*
@MONTH function *72*
mouse
debugging macros with *140*
MOVETO macro command *223*
moving
blocks *169*
pages *170*
to a block *188*
windows *276*
multiple rates of return *63*
multiplication *8*
Multiply.*Option* command equivalent *223*
multiplying
matrixes *223*

N

@N function *73*
@NA function *74*
NA values *9*
finding *66*
returning *74*
simple transactions and *62*
name
dialog box *303, 306*
SpeedBar *318*
NAME macro command *224*
NamedStyle.*Option* command equivalent *224*
naming
blocks *170*
natural logarithms, returning *68*
negation *8*
negative signs *16, 53*
nesting @functions *5*
net present value *76, 77*
New Dialog tool, Graphs page SpeedBar *307*
New SpeedBar tool, Graphs page SpeedBar *318*
NEXTPANE macro command *225*
NEXTWIN macro command *225*
non-blank cells, finding *32*

non-integer values *16*
Notebook.*Property* command equivalent *225*
notebooks
  active
    accessing from macro libraries *137*
  ANSI tables in *28*
  customizing
    macros *115*
  hidden comments in *6*
  linking *217*
  linking macros and *136*
  macro libraries *See* macro libraries
  macros *See* macros
  objects
    identifying *426*
    properties *426*
  printing *241*
  properties *225*
  scrolling *210, 211, 239, 273*
@NOW function *75*
@NPER function *75*
@NPV function *76*
Num Lock, turning on and off *227*
number, date/time serial
  date
    current *75, 100*
    day *37*
    month *72*
    specified *34, 35*
    year *105*
  time
    current *75*
    hour *56*
    minute *71*
    second *90*
    specified *99*
number argument (macros) *121*
numeric display format
  macro codes *177*
  specifying *176*
numeric display formats *94*
numeric values
  as arguments *6*
  averaging *24*
  choosing *29, 54, 103*
  converting hexadecimal to decimal *54*
  converting to hexadecimal *78*
  converting to strings *94, 101*
  database @functions and *12*
  entering
    in macro commands *121*
  macro command arguments and *121*
  mathematical @functions and *10*
  maximum *69*
  minimum *71*
  operator precedence and *8*
  prompting for *201*
  random *85*
  remainders, returning *72*
  returning *101*
  rounding off *89*
  square root *92*
  statistical @functions and *10*
  sum *95*
  system date *100*
  testing for *66, 73*
NUMOFF and NUMON macro commands *227*
@NUMTOHEX function *78*

O

Object Inspector menus
  controls *340*
  SpeedBars *321, 322*
object macro commands *123, 134, 145, 151, 159*
objects *See also* controls
  aligning *161*
  buttons *See* buttons
  creating *178*
  deleting *187*
  dialog box
    properties *408*
  floating *See* floating objects
  graph
    drawn
      properties *417*
    fixed
      properties *420*
  grouping in graphs *210*
  identifying *399*
  inserting *188*
  moving *223*
  notebook
    properties *426*

properties
inspecting 202, 203
setting 263, 264
resizing 255
running macros 115
selecting 260
offset values 59
OK button 303
creating 342
OLE links
creating 237
OLE objects
inserting 214
linking 229
OLE.*Option* command equivalent 229
ONERROR macro command 227
onscreen comments 6
OPEN macro command 229
opening
duplicate windows 276
files 192, 193
new files 192
workspaces 280
operators 7
arithmetic 8
compound conditions 57
logical 9
precedence 7
text 8
types 8
Optimizer.*Option* command equivalent 231
option buttons *See* radio buttons
order
controls 333
settings 330, 331
Order command, Dialog menu 326, 330
Order.*Option* command equivalent 233
ordering
floating objects 196
graph and dialog objects 233

P

page breaks
inserting 214
Page.*Property* command equivalent 233
pages
changing properties 233
grouping 210
in block 91
inserting 169
moving 170
scrolling
horizontally 211
vertically 239, 273
PANELOFF macro command 201, 236
PANELON macro command 236
panes
switching 225
parent control 335
moving 336-338
parentheses
@function arguments and 5
overriding operator precedence 8
Parse.*Option* command equivalent 237
parsing
data strings 237
PasteFormat command equivalent 237
PasteLink command equivalent 238
PasteSpecial command equivalent 238
pasting
from the clipboard 188
PAUSEMACRO macro command 239
@PAYMT function 78
performance
macro speed 236, 278
PGUP and PGDN macro commands 239
pi, returning 79
@PI function 79
pick list 348
@PMT function 79
pointing
@functions and 5
POKE macro command 240
polygon
creating 178
polyline
creating 178
population statistics 12
@functions returning 11
standard deviation 44, 93
variance 47, 102
positive signs 16, 53

@PPAYMT function *81*
precedence of operators *7*
present value
  calculating *83, 84*
  net *76, 77*
Preview command equivalent *240*
primary notebooks
  startup macros *115*
principal
  future value *78*
  vs. interest portion of payment *60, 81*
Print.*Option* command equivalent *241*
print preview *240*
PrinterSetup command equivalent *243*
printing
  data and graphs *241*
  printer setup *243*
program flow macro commands *122, 125, 145, 148*
Program Manager
  running macros from *112*
prompts
  clearing *172*
  disabling *236*
  enabling *236*
  for labels *200*
  for numeric values *201*
  running macros from *112*
proper case, changing to *82*
@PROPER function *82*
properties
  application *162*
  buttons *340*
  changing *311, 322*
    with macro commands *296*
  combo boxes *348*
  controls *324, 334-335, 340*
  dialog box object
    identifying *408*
  dialog window *182*
  edit fields *344-345*
  graph *207*
  graph object (drawn)
    identifying *417*
  graph object (fixed)
    identifying *420*
  graph window *209*
  inspecting *202, 203*
  list boxes *347*
  menu commands *380*
  notebook *225*
    identifying *426*
  page *233*
  rectangles *346*
  returning current settings *82*
  scroll bars *350*
  setting *263, 264*
  SpeedBars *322*
  spin controls *345*
@PROPERTY function *82*
punctuation
  in macro arguments *120*
push buttons *See also* buttons, *307*
  bitmap *342*
  creating *319, 340*
  properties *340*
PUT macro command *244*
PUTBLOCK macro command *245*
PUTCELL macro command *246*
@PV function *83*
@PVAL function *84*

Q

QGOTO macro command *246*
Quattro Pro for DOS
  macros
    compatibility with *116, 146*
QUERY macro command *246*
Query.*Option* command equivalent *247*
question mark (?)
  in macros *130*
  macro command *157*
QUIKSAVE.WB1 *372*
QUIT macro command *111, 117, 247*
quote marks (")
  in macros *121*

R

R macro command *256*
radians *10*
  converting to degrees *41*
@RADIANS function *85*
radio buttons *See also* buttons, *307, 333*

bitmap *342*
creating *309, 341*
group boxes *309, 346*
group boxes and *341*
properties *340*
@RAND function *85*
random numbers, generating *85*
ranges *See* blocks
rate of return
internal *62*
multiple *63*
@RATE function *86*
READ macro command *248*
reading files *230*
READLN macro command *249*
REC indicator *108*
RECALC macro command *250*
RECALCCOL macro command *251*
recalculation *173*
columnwise order *251*
row-by-row order *250*
RECEIVE link command *360, 377*
recording macros *108-110*
rectangle
creating *178*
dialog box *308, 335, 346*
redrawing *See* updating
references
absolute cell address *110*
block
macro command arguments *122*
macro command arguments and *121*
indirect *185*
macro libraries and *136*
number argument in macro commands *121*
relative *137*
reformatting
blocks *171*
Regression.*Option* command equivalent *252*
remainders, returning *72*
@REPEAT function *87*
repeating characters *87*
repeating label prefix (\) *87*
@REPLACE function *87*
REQUEST macro command *253*
RESIZE macro command *254*
ResizeToSame command equivalent *255*
resizing
windows *277, 278*
RESTART macro command *255*
RestrictInput.*Option* command equivalent *255*
retrieving
files *193*
RETURN macro command *117, 125, 256*
right-clicking to display Object Inspector menus *322, 340*
RIGHT macro command *256*
@RIGHT function *88*
@ROUND function *89*
rounding off numbers *89*
row-by-row recalculation *250*
ROWCOLSHOW macro command *256*
ROWHEIGHT macro command *257*
rows
down *187*
hiding *256*
in block *89*
inserting *169*
resizing *257*
return value *103*
revealing *256*
totaling *268*
up *272*
@ROWS function *89*
running macros *112, 115*
changing macro while *138*
in Debug mode *140*
in different notebooks *136*

S

@S function *90*
sample statistics *12*
@functions returning *11*
standard deviation *45, 94*
variance *48, 103*
Save SpeedBar As command
Dialog menu *318*
saving
dialog boxes *307*
files *193*
SpeedBars *318*
workspaces *280*
scalar (dot) product *96*

screen macro commands *122, 124, 145, 147*
screens
  updating *236, 279*
    disabling *112, 236, 278*
scripts *See* macros
scroll bars
  dialog boxes *350*
scroll lock
  turning on and off *258*
scrolling notebooks *210, 211, 239, 273*
SCROLLON and SCROLLOFF macro commands *258*
Search key (F2) *123*
Search.*Option* command equivalent *258*
@SECOND function *90*
Securities Industries Association Handbook *16*
Select key (Shift+F7)
  macro equivalent *219*
SELECTBLOCK macro command *259*
SELECTFLOAT macro command *259*
selecting
  blocks *188, 259*
  controls *305, 324*
  floating objects *259*
  objects *260*
Selection tool, dialog window SpeedBar *305, 324*
SELECTOBJECT macro command *260*
selector (cell) *See* cell selector
semicolon (;)
  in comments *6, 156*
  macro command *156*
SEND link command *357, 377*
sending controls backward *326*
separators
  in @function statements *5*
sequential definition (arguments) *180*
Series.*Option* command equivalent *260*
server, DDE conversation *131*
SET link command *357, 377*
SETGRAPHATTR macro command *262*
SETMENUBAR macro command *262*
SETOBJECTPROPERTY macro command *263*
  hiding menus *379*
  shortcut keys *378*
SETPOS macro command *264*
SETPROPERTY macro command *264*
settings, dialog box *329*
  ordering *330, 331*
@SHEETS function *91*
SHIFT macro command *265*
Short International Time Format *100*
shortcut keys *372*
  assigning *378*
  dialog boxes *332*
  specifying *375*
simple transactions *62*
@SIN function *91*
sine, returning *91*
Slide.*Option* command equivalent *265*
slide shows
  running *265*
@SLN function *92*
SolveFor.*Option* command equivalent *266*
solving
  "what-if" problems *274*
  equation sets *231*
  goal-seeking problems *231, 266*
  inequalities *231*
  matrix problems *215, 223*
  regression analysis *252*
Sort.*Option* command equivalent *267*
sorting
  data *267*
spacebar
  macros and *140, 141*
SpeedBar
  entering @functions with *4*
SpeedBars *302*
  Button tool *113*
  creating *293, 302-306, 318-322*
  customizing *322*
  dialog window *304, 307*
  displaying *321*
  editing *322*
  Graphs page *307, 318*
  names *318*
  saving *302, 318*
  testing *322, 323*
  text, adding *319, 331*
  UI Builder *See* UI Builder
SpeedButtons *113*
SpeedFill button
  macro equivalent *267*

SPEEDFILL macro command 267
SPEEDFORMAT macro command 268
SpeedSum button
  macro equivalent 268
SPEEDSUM macro command 268
spin control 308, 345
@SQRT function 92
square roots 92
stack 127
standard breakpoints 140, 141
  conditional vs. 140
  removing 144
standard deviation 11
  population statistics 44, 93
  sample statistics 45, 94
standard time formats 99
  vs. military 56
startup options
  autoload macros 115
statistical @functions 10, 11
  formulas 11
  labels and 11
  vs. database @functions 12
status line
  hints 324, 372
  updating 201, 236
@STD function 93
@STDS function 94
STEP macro command 268
STEPOFF macro command 269
STEPON macro command 269
straight-line depreciation allowance 92
string @functions 18
@STRING function 94
string values
  as arguments 6
  characters
    extracting 70
      left 67
      right 88
    finding number 68
    replacing 87
  choosing 29, 54, 103
  comparing 49
  extra spaces in, removing 100
  hexadecimal 78
  index 51
  lowercase, changing to 69
  proper case, changing to 82
  returning 90
  testing for 67
  text formulas and 8
  time 99
  uppercase, changing to 101
  words in 82
strings
  argument type in
    macros 121
  combining with ampersands 8, 68
  comparing 49
  converting numeric values to 94
  converting to ANSI characters 30
  converting to numeric values 101
  empty 70, 89, 90
  inserting CR/LF characters in 281
  relational operators and 8
styles
  creating and using 224
submenus *See also* menus, 371
  deleting 379
Subroutine macro command 269
subroutines 125, 126, 269
  arguments and 180
  calling 269
  canceling 198
  running 197
  terminating 255, 256
subtraction 8
@SUM function 95
@SUMPRODUCT function 96
supporting notebooks
  startup macros 115
@SYD function 96
syntax
  command definition 374
  error 120, 121
  @functions 4-9
  macros 120
system beeps 166

T

Tab key
  dialog box, setting 333
  macro equivalent 167, 270

TAB macro command *270*
TABLE macro command *271*
TableQuery.*Option* command equivalent *271*
tables
  ANSI *28*
  index *58*
  lookup *73*
  searching horizontally *54*
  searching vertically *104*
TableView command equivalent *271*
@TAN function *97*
tangents *97*
@TERM function *98*
TERMINATE macro command *272*
Test button, dialog window SpeedBar *304, 317*
Test mode *304, 317, 323*
testing
  dialog boxes *304, 317, 323*
  SpeedBars *322, 323*
text
  buttons *319, 331*
  controls *331*
  dialog boxes *311, 331, 342-344*
  object *178*
  SpeedBars *319, 331*
text files *230*
text formulas
  macros and *117, 121, 122, 138*
text operators *8*
text strings
  macros and *121*
tilde (~)
  command *155, 178*
tiling
  windows into panes *279*
time
  current, returning *75*
  displaying *351-352*
  formats *17, 99*
    international *100*
  hour, returning *56*
  macro command *179*
  minute, returning *71*
  second, returning *90*
  serial values, returning *99*
time @functions *17*
time control *351-352, 364*
@TIME function *99*
timers *351-352, 364*
@TIMEVALUE function *99*
title
  dialog box *304, 307*
titles
  locking *279*
@TODAY function *100*
tools *See also* specific types, *304, 307*
Tools menu
  Macro *108*
  UI Builder *303*
topic, DDE conversation *132*
trace cells *143*
  removing *144*
trace window *140*
transposing
  blocks *171*
trapping events *353-355*
TRIGGER link command *363*
@TRIM function *100*
@TRUE function *101*
Truth in Lending Law *16*
type-ahead buffer, checking *199, 217*
typing macros *117-136*

U

UI Builder *303*
  SpeedBar *304*
  tools *304, 307*
    Bitmap Button *341*
    Check Box *333*
    Color Control *349*
    Combo Box *347*
    File Control *349*
    Group Box *346*
    Group box *309*
    Horizontal Scroll Bar *350*
    Label *311, 331, 342-344*
    List Box *347*
    Pick List *348*
    Push Button *340*
    Radio Button *309, 333, 341*
    Rectangle *335*
    Timer *351-352, 364*
    Vertical Scroll Bar *350*
  using *306*

UI building macro commands *123, 133, 145, 150*
underlined letters
  dialog boxes *332*
  menus *372*
UNDO macro command *272*
UngroupObjects command equivalent *272*
UP macro command *272*
updating
  input line *201, 236*
  menus *201, 236*
  screens *236, 278, 279*
  status line *201, 236*
  suppressing *112*
@UPPER function *101*
uppercase characters *101*
  changing to lowercase *69*

V

value entry vs. label entry
  in macros *122*
@VALUE function *101*
values
  absolute *22*
  converting formulas to *171*
  entering *216, 244, 246*
    same repeated *245*
  fields
    averaging *36*
    maximum, finding *42*
    minimum, finding *43*
    totaling *46*
  index *27, 55, 58, 104*
  logical true/false *211*
    macros and *122, 142*
  non-integer *16*
  offset *59*
  random numbers *85*
values, date/time serial
  date
    current *75, 100*
    day *37*
    month *72*
    specified *34, 35*
    year *105*
  time
    current *75*
    hour *56*
    minute *71*
    second *90*
    specified *99*
values, serial date/time *17*
  time *17*
@VAR function *102*
variance *11*
  population statistics *47, 102*
  sample statistics *48, 103*
@VARS function *103*
@VERSION function *103*
vertical scroll bars *350*
viewing
  dialog windows *182*
  graphs *208*
VLINE macro command *273*
@VLOOKUP function *103*
VPAGE macro command *273*

W

WAIT indicator *273*
WAIT macro command *273*
warning beeps *166*
WhatIf.*Option* command equivalents *274*
WINDOW macro command *275*
WINDOW<Number> macro command *275*
WindowArrIcon command equivalent *275*
WindowCascade command equivalent *275*
WindowClose command equivalent *275*
WindowHide command equivalent *276*
WindowMaximize command equivalent *276*
WindowMinimize command equivalent *276*
WindowMove command equivalent *276*
WindowNewView command equivalent *276*
WindowNext command equivalent *277*
WindowPanes command equivalent *277*
WindowRestore command equivalent *277*
windows
  activating *158*
  arranging icons *275*
  cascading *275*
  closing *275*
  debug *139, 140, 144*

dialog *See* dialog window
displaying hidden *277*
displaying next *277*
dividing into panes *277*
hiding *276*
listing open *173, 204*
locking titles *279*
macro *139*
maximizing *276*
minimizing *276*
moving *276*
objects
  creating *178*
  selecting *260*
opening duplicate *276*
opening new *192*
panes
  switching *225*
resizing *278*
restoring minimized *277*
scrolling *210, 211, 239, 273*
switching *275*
tiling *279*
trace *140*
updating
  disabling *112*
viewing dialog *182*
WindowShow command equivalent *277*
WindowSize command equivalent *278*
WINDOWSOFF macro command *278*
WINDOWSON macro command *279*
WindowTile command equivalent *279*
WindowTitles command equivalent *279*
words
  in string values, defined *82*
WorkSpace.*Option* command equivalent *280*
WRITE macro command *280*
WRITELN macro command *281*
writing to files *230*

X

/x macro commands *123, 155*

Y

@YEAR function *105*

Z

ZOOM macro command *282*

QUATTRO® PRO
FOR WINDOWS

B O R L A N D

CORPORATE HEADQUARTERS: 1800 GREEN HILLS ROAD, P.O. BOX 660001, SCOTTS VALLEY, CA 95067-0001, (408) 438-5300. OFFICES IN: AUSTRALIA, BELGIUM, CANADA, DENMARK, FRANCE, GERMANY, HONG KONG, ITALY, JAPAN, KOREA, MALAYSIA, THE NETHERLANDS, NEW ZEALAND, SINGAPORE, SPAIN, SWEDEN, TAIWAN AND THE UNITED KINGDOM ■ PART #24MN-QPW04-10 ■ BOR 2942