

## To translate a Quattro Pro macro command to PerfectScript Syntax,

- 1 Remove the opening brace of the Quattro Pro macro command and place it just after command name. For example, change {BlockCopy A1..A20,C10} to BlockCopy{A1..A20,C10}. For commands without arguments, move the opening brace to just before the closing brace. For example, change {DOWN} to DOWN{ }.
- 2 Replace the braces with parentheses. For example, change DOWN{ } to DOWN() and BlockCopy{A1..A20,C10} to BlockCopy(A1..A20,C10).
- 3 Change any periods in the command name to underscores. For example, change Print.Block(A1..A20) to Print\_Block(A1..A20).
- 4 Change commas that separate arguments to semicolons.
- 5 If an argument is calculated using a formula, enclose the formula in an Eval() statement. For example, change BlockFill\_Start(+A20\*0.25) to BlockFill\_Start(Eval("+A20\*0.25")). Eval() works only in PerfectScript macros.



### Note

- When you use the Quattro Pro commands in PerfectScript, you need to use slightly different syntax than in Quattro Pro. You can translate Quattro Pro commands to PerfectScript syntax. Or, you can record a macro as a PerfectScript macro, then access a list of PerfectScript commands to edit the macro.
- Now you can copy these commands into the application you are using to edit PerfectScript macros, such as WordPerfect.
- When you convert Quattro Pro macros to PerfectScript macros, the following Quattro Pro macro commands must be replaced by the corresponding PerfectScript commands:

<b>Quattro Pro</b>	<b>PerfectScript</b>
{ABS}	AbsoluteReference()
{ASSIGN}	AssignValue()

## Keyboard Macro Commands

When you record a macro, Quattro Pro translates special keystrokes into keyboard commands. When you type in a macro, use these commands in place of keystrokes. There are four categories:

- ◆ Function Keys  
F1 through F12
- ◆ Status Keys  
Caps Lock, Scroll Lock, and other indicator keys
- ◆ Other Keys  
Break, Esc, Del, plus other keys and key combinations

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}~{DOWN}
```

The following macro deletes the last three characters of whatever is in the active cell:

```
{EDIT}{END}{SHIFT+LEFT 3}{DEL}{CR}
```

For compatibility with future versions of Quattro Pro, use command equivalents to execute menu items instead of using keyboard macros.

 **Related topics**

## Movement Key Macro Commands

Use these macro commands in place of arrows and other movement keys:

<u>{HOME}</u>	Home
<u>{END}</u>	End
<u>{LEFT}</u> or <u>{L}</u>	Left arrow
<u>{RIGHT}</u> or <u>{R}</u>	Right arrow
<u>{UP}</u> or <u>{U}</u>	Up arrow
<u>{DOWN}</u> or <u>{D}</u>	Down arrow
<u>{PGUP}</u>	PgUp
<u>{PGDN}</u>	PgDn
<u>{BIGLEFT}</u> or <u>{BACKTAB}</u>	Ctrl+Left arrow or Shift+Tab
<u>{BIGRIGHT}</u> or <u>{TAB}</u>	Ctrl+Right arrow or Tab

 **Related topics**

## Function Key Macro Commands

These macro commands are equivalent to Quattro Pro function keys and key combinations:

<u>{CALC}</u>	F9
<u>{EDIT}</u>	F2
<u>{FUNCTIONS}</u>	Alt+F3
<u>{GRAPH}</u>	F11
<u>{HELP}</u>	F1
<u>{INSPECT}</u>	F12
<u>{MACROS}</u>	Shift+F3
<u>{MARK}</u>	Shift+F7
<u>{NAME}</u>	F3
<u>{NEXTPANE}</u>	F6
<u>{NEXTTOPWIN}</u>	Ctrl+F6
<u>{NEXTWIN}</u>	Shift+F6
<u>{OBJECTSPAGEGO TO}</u>	Shift+F5
<u>{QUERY}</u>	F7
<u>{TABLE}</u>	F8



**Related topics**

## Status Key Macro Commands

These macro commands are equivalent to the status indicator keys:

<u>{CAPOFF}</u>	Caps Lock off
<u>{CAPON}</u>	Caps Lock on
<u>{INSOFF}</u>	Ins off
<u>{INSON}</u>	Ins on
<u>{NUMOFF}</u>	Num Lock off
<u>{NUMON}</u>	Num Lock on
<u>{SCROLLOFF}</u>	Scroll Lock off
<u>{SCROLLON}</u>	Scroll Lock on



**Related topics**

## Macro Commands for Other Keys

These macro commands are used for building key combinations and for entering or editing data:

{CLEAR}

Ctrl+Backspace (Clears any existing entry from a prompt line or from the input line in Edit mode)

{CR} or ~

Enter

{DATE}

Ctrl+Shift+D

{SHIFT}

The keystroke Key while Shift is held down



**Related topics**

## Screen Macro Commands

These macro commands affect the display:

### {BEEP}

Sounds the computer's bell.

### {INDICATE}

Sets the mode indicator to display a specified string.

### {PANELOFF}

Suppresses menus and prompts.

### {PANELON}

Restores menus and prompts disabled by {PANELOFF}.

### {WINDOWSOFF}

Prevents screen updating.

### {WINDOWSON}

Restores screen updating disabled by {WINDOWSOFF}.

Screen commands can control the status line, suppress screen redraws, or sound the computer's bell. For example, {INDICATE "WAIT"} changes the status line indicator to WAIT; {INDICATE} resets the status line to its default.

## Cell Macro Commands

These macro commands affect the data stored in specific cells. They can enter and read data, recalculate specific cells, or convert values into formatted labels:

### {BLANK}

Erases a cell or cells.

### {CONTENTS}

Copies the contents of one cell to another as a formatted label.

### {GETDIRECTORYCONTENTS}

Enters an alphabetized list of files in a specified directory into specified cells.

### {GETWINDOWLIST}

Lists open windows in specified cells.

### {LET}

Places a number or string in the given location.

### {NAVIGATE.SelectTable}

Expands selection of table boundaries.

### {NAVIGATE.Zoom2Fit}

Zooms a table so that it fits in the visible part of the screen.

### {NAVIGATE.GoTo}

Navigates to the sides or corners of a table.

### {NAVIGATE.Jump}

Jumps to the next table in a given direction, or to the current table boundary if currently within a table.

### {PUT}

Places a number or string in a given location offset by the specified number of columns and rows.

### {PUTBLOCK}

Stores a value in each specified cell.

### {PUTBLOCK2}

Stores Data in each cell of Block; same as {PUTBLOCK} except it parses date formats and requires a formula prefix before values.

### {PUTCELL}

Stores the specified data in the active cell.

### {PUTCELL2}

Stores the specified data in the active cell; same as {PUTCELL} except it parses date formats and requires a formula prefix before values.

### {RECALC}

Recalculates formulas in the specified location in row-wise order for a specified number of times, or until a specified condition is reached.

### {RECALCCOL}

Recalculates formulas in the specified location in column-wise order for a specified number of times, or until a specified condition is reached.

### {SPEEDFILL}

Fills the selected cells with sequential data.

### {SPEEDFORMAT}

Applies a standard format to the selected cells.

### {SPEEDFORMAT.Add}

Adds a new format based on a specified cells.

### {SPEEDFORMAT.Remove}

Removes a format.

### {SPEEDSUM}

Adds filled cells in the specified cells and writes the sum in adjacent empty cells (right/bottom).



## Interactive Macro Commands

Interactive commands let you create macros that accept users' input. These commands can display dialog boxes and pop-up menus, pause the macro and request input, and prevent you from breaking out of the macro during a crucial period:

{ ? }

Pauses the macro and accepts input from the keyboard until you press Enter.

{ACTIVATE}

Makes the specified window active.

{DODIALOG}

Displays the specified dialog box.

{GET}

Pauses the macro, accepts one keystroke, and stores it in the specified location.

{GETLABEL}

Pauses the macro, displays a specified prompt, and stores the subsequent keystrokes as a label in the specified location.

{GETNUMBER}

Pauses the macro, displays the specified prompt, and stores the subsequent keystrokes as a numeric value in the specified location.

{PAUSEMACRO}

Pauses the macro for dialog box entries, then resumes the macro when the user closes the dialog box.

{UNDO}

"Takes back" the last command; cancels its effects.

{WINDOW}

Makes the next pane active.

{WINDOW}

Makes the specified window active.

## Program Flow Macro Commands

Program flow commands let you branch and loop in a macro. Use these commands to create macros that behave differently based on logical decisions, to break down complicated macros, or to consolidate repetitious macros into subroutines:

### {DEFINE}

Defines the type of arguments passed to a subroutine and tells where to store them.

### {FORBREAK}

Terminates execution of a {FOR} command.

### {QUIT}

Terminates macro execution and returns keyboard control.

## File Macro Commands

File macro commands work with data in files other than the active notebook. These commands can open files and read their data character by character, or create custom data files. 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. Use this feature to trap errors such as the disk being full or reading data past the end of the file:

### {ANSIREAD}

Reads the specified number of bytes and stores them in the specified location, without any character mapping.

### {ANSIREADLN}

Reads a line of characters and stores it in the specified location, without any character mapping.

### {ANSIWRITE}

Writes a string of characters to the current file, without any character mapping.

### {ANSIWRITELN}

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}

Calculates the number of bytes in the current file, and stores the value in the specified location.

### {GETPOS}

Places the position of the file pointer in the specified location.

### {OPEN}

Opens a file for reading, writing, modifying, or appending.

### {READ}

Reads the specified number of bytes and stores them in the specified location.

### {READLN}

Reads a line of characters and stores it in the specified location.

### {SETPOS}

Sets the file pointer to the specified byte position in a file.

### {WRITE}

Writes a string of characters to the current file.

### {WRITELN}

Writes a string of characters to the current file and ends it with a carriage return/linefeed.

## **OLE and DDE Macro Commands**

OLE and DDE macro commands communicate with other Windows applications:

### {DELVAR}

Deletes the specified named variable and frees any object assigned to that variable.

### {EXECAUTO}

Evaluates one or more automation expressions, but drops any returned values from expressions.

### {EXECUTE}

Runs the specified macro in a server application.

### {POKE}

Sends data to a DDE application.

### {REQUEST}

Requests data from a DDE application.

### {SETLCID}

Specifies a locale ID for automation expressions.

## UI Building Macro Commands

UI building commands can add or remove menu items from the active menu bar:

### {ADDMENU}

Adds a menu to the menu system at a specified location.

### {ADDMENUITEM}

Adds a menu item to the menu system at a specified location.

### {ADDSUBMENUITEM}

Adds a menu item to the menu system at a specified location and makes the command at that location a menu.

### {DELETEMENU}

Deletes a menu from the active menu system.

### {DELETEMENUITEM}

Deletes a menu item from the active menu system.

### {SETMENUBAR}

Sets the active menu system.

## Object Macro Commands

Object commands let you move and resize objects on the Objects sheet either in a chart window, in a notebook window, or in a dialog window. These commands can also create objects or change the property settings of Quattro Pro objects. Object commands can affect objects in any Quattro Pro window.

### {ADDSERIES}

Adds a data series to a floating chart.

### {COLUMNWIDTH}

Sets the width of columns in the specified cells.

### {CREATEOBJECT}

Creates chart or dialog objects.

### {DUPLICATE}

Duplicates selected chart annotations or dialog box controls.

### {FLOATCOPY}

Copies a floating object in the active notebook window.

### {FLOATCREATE}

Creates a floating chart, SpeedButton, or drawn object.

### {FLOATMOVE}

Moves a floating object in the active notebook window.

### {FLOATSIZE}

Resizes a selected floating object in the active notebook window.

### {FLOATTEXT}

Replaces the string in the selected floating text box with the specified string.

### {GETOBJECTPROPERTY}

Gets property setting from an object.

### {GETPROPERTY}

Gets property setting from the object currently selected.

### {MOVETO}

Moves selected objects to coordinates x, y.

### {RESIZE}

Resizes selected objects in the active window.

### {ROWCOLSHOW}

Shows or hides rows or columns in the specified cells.

### {ROWHEIGHT}

Sets the height of rows in the specified cells.

### {SELECTBLOCK}

Selects a contiguous or noncontiguous selection within the active notebook.

### {SELECTFLOAT}

Selects floating objects in the active notebook window using their object names.

### {SELECTOBJECT}

Selects objects in active window.

### {SETGRAPHATTR}

Equivalent to setting Chart Setup and Background properties (colors are comma-separated RGB triplets in quotes).

### {SETOBJECTPROPERTY}

Sets properties of objects.

### {SETPROPERTY}

Sets properties of selected object.

## Miscellaneous Macro Commands

These commands insert characters and comments, and perform a variety of other tasks. The slide commands at the end of the list are used to build and run slide shows:

<u>} or { }</u>	Inserts a right brace, }
<u>{ }</u>	Inserts a left brace, {
<u>{ ~ }</u>	Inserts a tilde, ~
<u>{ }</u> (blank)	Does nothing; lets you reserve space in a macro without stopping macro execution.
<u>{ : }</u> (semicolon)	Lets you enter explanatory remarks in a macro.
<u>{ HLINE }</u>	Scrolls window horizontally the specified number of lines.
<u>{ HPAGE }</u>	Scrolls window horizontally the specified number of pages.
<u>{ VLINE }</u>	Scrolls window vertically the specified number of lines.
<u>{ VPAGE }</u>	Scrolls window vertically the specified number of pages.

For slide show commands, see { Slide.Option }

## Command Equivalent Macros

Use the following command equivalent macros in place of menu items. When used with @COMMAND, they return current command settings.

### {Align.Option}

Equivalent to Format ▶ Align.

### {AnalysisExpert}

Equivalent to Tools ▶ Numeric Tools  
▶ Analysis.

### {Application.Property}

Equivalent to Tools ▶ Settings.

### {BlockCopy}

Equivalent to Edit ▶ Copy Cells.

### {BlockDelete.Option}

Equivalent to Edit ▶ Delete.

### {BlockFill.Option}

Equivalent to Edit ▶ Fill  
▶ Fill Series.

### {BlockInsert.Option}

Equivalent to Insert ▶ Cells.

### {BlockMovePages}

Equivalent to Edit ▶ Move Sheets.

### {BlockName.Option}

Equivalent to Insert ▶ Cell Names.

### {BlockReformat}

Equivalent to Format ▶ Text Reformat.

### {BlockTranspose}

Equivalent to Tools ▶ Numeric Tools  
▶ Transpose.

### {BlockValues}

Equivalent to Edit ▶ Convert to Values.

### {BudgetExpert}

Equivalent to Tools ▶ Numeric Tools  
▶ Budget.

### {ChartExpert}

Equivalent to Insert ▶ Chart.


### {ClearContents}

Equivalent to Edit ▶ Clear  
▶ Values.

### {ClearFormats}

Equivalent to Edit ▶ Clear  
▶ Formats.

### {ComposeFormula}

Equivalent to clicking .

### {Consolidate.Option}

Equivalent to Tools ▶ Consolidate  
▶ Edit.

### {ConsolidateExpert}

Equivalent to Tools ▶ Consolidate  
▶ New.

### {Controls.Option}

Equivalent to Format ▶ Object Order.



{DbAlias}

Equivalent to Insert ► External Data  
► Aliases.

{DialogView}

Equivalent to Tools ► Macro  
► Dialog Designer.

{DialogWindow.Property}

Equivalent to right-clicking the title bar of a custom dialog window.

{EditClear}

Equivalent to Edit ► Clear.

{EditCopy}

Equivalent to Edit ► Copy.

{EditCut}

Equivalent to Edit ► Cut.

{EditGoto}

Equivalent to Edit ► Go To.

{EditPaste}

Equivalent to Edit ► Paste.

{ExportGraphic}

Equivalent to Chart ► Export to File.

{FileClose}

Equivalent to File ► Close.

{FileCloseAll}

No equivalent command.

{FileCombine}

Equivalent to Tools ► Data Tools  
► Combine Files.

{FileExit}

Equivalent to File ► Exit.

{FileExtract}

Equivalent to Tools ► Data Tools  
► Extract to File.

{FileImport}

Equivalent to Tools ► Data Tools  
► QuickColumns.

{FileNew}

Equivalent to File ► New.

{FileOpen}

Equivalent to File ► Open.

{FileRetrieve}

No equivalent command.

{FileSave}

Equivalent to File ► Save.

{FileSaveAll}

No equivalent command.

{FileSaveAs}

Equivalent to File ► Save As.

{FileSend}

Equivalent to File ► Send To  
► Mail.

{FloatOrder.Option}

Equivalent to Format ► Object Order.

{Frequency.Option}

Equivalent to Tools ▶ Numeric Tools  
▶ Frequency.

{GraphCopy}  
Equivalent to Edit ▶ Paste Special.

{GraphDeactivate}  
No equivalent command.

{GraphDelete}  
Equivalent to Chart ▶ Delete Chart.

{GraphEdit}  
Equivalent to right-clicking and clicking Edit Chart.

{GraphGallery}  
Equivalent to Chart ▶ Gallery.

{GraphNew}  
Equivalent to Chart ▶ New Chart.

{GraphSettings.Titles}  
Equivalent to Chart ▶ Titles.

{GraphSettings.Type}  
Equivalent to Chart ▶ Type/Layout.

{GraphView}  
Equivalent to Chart ▶ View Chart.

{GraphWindow.Property}  
Equivalent to right-clicking the title bar of a chart window.

{Group.Option}  
Equivalent to Insert ▶ Name  
▶ Groups of Sheets.

{GroupObjects}  
Equivalent to Format ▶ Group.

{ImportGraphic}  
Equivalent to Insert ▶ Graphics  
▶ Clipart.

{InsertBreak}  
Equivalent to Insert ▶ Page Break.

{InsertObject}  
Equivalent to Insert ▶ Object.

{Invert.Option}  
Equivalent to Tools ▶ Numeric Tools  
▶ Invert.

{Links.Option}  
Equivalent to Edit ▶ Links.

{MapExpert}  
Equivalent to Insert ▶ Graphics  
▶ Map.

{Multiply.Option}  
Equivalent to Tools ▶ Numeric Tools  
▶ Multiply.

{NamedStyle.Option}  
Equivalent to Format ▶ Styles.

{Notebook.Property}  
Equivalent to Format ▶ Notebook.

{Notebook.Summary.Option}  
Equivalent to File ▶ Properties.

{ObjectsPageGoto}  
Equivalent to View ▶ Objects Sheet.

{OLE.Option}

Equivalent to right-clicking an OLE object and clicking the Property command.

{Optimizer.Option}

Equivalent to Tools ▶ Numeric Tools

▶ Optimizer.

{Order.Option}

Equivalent to Format ▶ Object Order.

{Page.Property}

Equivalent to Format ▶ Sheet.

{Parse.Option}

Equivalent to Tools ▶ Data Tools

▶ QuickColumns.

{PasteFormat}

Equivalent to Edit ▶ Paste Special.

{PasteLink}

Equivalent to Edit ▶ Paste Special.

{PasteSpecial}

Equivalent to Edit ▶ Paste Special.

{PlayPerfectScript}

Equivalent to Tools ▶ Macro

▶ Play.

{Preview}

Equivalent to File ▶ Print Preview.

{Print.Option}

Equivalent to File ▶ Page Setup.

{PrinterSetup}

Equivalent to File ▶ Print, then specifying a printer.

{Query.Option}

Equivalent to Tools ▶ Data Tools

▶ Notebook Query.

{Regression.Option}

Equivalent to Tools ▶ Numeric Tools

▶ Regression.

{ResizeToSame}

Equivalent to Format ▶ ResizeToSame.

{RestrictInput.Option}

No equivalent command.

{Scenario.Option}

Equivalent to Tools ▶ Scenario

▶ Edit.

{ScenarioExpert}

Equivalent to Tools ▶ Scenario

▶ New.

{Search.Option}

Equivalent to Edit ▶ Find And Replace.

{SelectAll}

Equivalent to Edit ▶ Select All.

{Series.Option}

Equivalent to Chart ▶ Series.

{SeriesManager.Option}

Equivalent to Edit ▶ Define QuickFill; Series.

{Slide.Run}

Equivalent to Slides ▶ Play Slide Show.

{SlideShowExpert}

Equivalent to Tools ▶ Slide Show  
▶ New.

{SolveFor.Option}

Equivalent to Tools ▶ Numeric Tools  
▶ Solve For.

{Sort.Option}

Equivalent to Tools ▶ Sort.

{TableQuery.Option}

Equivalent to Insert ▶ External Data  
▶ Table Query.

{TableLink.Option}

Equivalent to Insert ▶ External Data  
▶ Table Link.

{TableView}

Equivalent to Insert ▶ External Data  
▶ Database Desktop.

{Toolbar.Option}

Equivalent to View ▶ Toolbars.

{UngroupObjects}

Equivalent to Format ▶ Ungroup.

{WhatIf.Option}

Equivalent to Tools ▶ Numeric Tools  
▶ What-If.

{WhatIfExpert}

Equivalent to Tools ▶ Numeric Tools  
▶ What-If.

{WindowArrIcon}

Equivalent to Window ▶ Arrange Icons.

{WindowCascade}

Equivalent to Window ▶ Cascade.

{WindowClose}

Equivalent to Control Menu ▶ Close.

{WindowHide}

Equivalent to Window ▶ Hide.

{WindowMaximize}

Equivalent to Control Menu ▶ Maximize.

{WindowMinimize}

Equivalent to Control Menu ▶ Minimize.

{WindowMove}

Equivalent to Control Menu ▶ Move.

{WindowNewView}

Equivalent to Window ▶ New View.

{WindowNext}

Equivalent to Control Menu ▶ Next.

{WindowPanels}

Equivalent to View ▶ Split Window.

{WindowQPW.Option}

Equivalent to Quattro Pro Control Menu.

{WindowRestore}

Equivalent to Control Menu ▶ Restore.

{WindowShow}

Equivalent to Window ▶ Show.

{WindowSize}

Equivalent to Control Menu ▶ Size.

{WindowTile}

Equivalent to Window ▶ Tile.

{WindowTitles}

Equivalent to View ▶ Locked Titles.

{WorkSpace.Option}

Equivalent to File ▶ Workspace.

## Analysis Tools Macro Commands

Analysis Tools commands let you perform analyses of numerical data. They are equivalent to the tools available when you choose Tools ► Numeric Tools

► Analysis.

{ANOVA1}

One-way analysis of variance.

{ANOVA2}

Two-way analysis of variance with replication.

{ANOVA3}

Two-way analysis of variance without replication.

{DESCR}

Creates a table of descriptive statistics that characterize one or more samples.

{EXPON}

Exponentially smooths a series of values.

{FOURIER}

Performs a fast Fourier transformation on cells of data.

{FTESTV}

Performs a two-sample F-Test to compare population variances.

{HISTOGRAM}

Calculates the probability and cumulative distributions for a sample based on a series of bins.

{MCORREL}

Returns the correlation matrix between two or more data sets.

{MCOVAR}

Returns the covariance matrix between two or more data sets.

{MOVEAVG}

Returns a moving average for the values in a sample.

{MTGAMT}

Generates an amortization schedule for a mortgage.

{MTGREFI}

Generates a table of information relating to refinancing a mortgage.

{PTTESTM}

Performs a paired two-sample Student's t-Test for means.

{PTTESTV}

Performs a Student's t-Test using two independent samples with unequal variances.

{RANDOM}

Returns random values from a specified distribution.

{RANKPERC}

Computes the ordinal and percent rank of each value in a sample.

{REGRESS}

Performs multiple linear regression analysis.

{SAMPLE}

Returns a periodic or random sample of values from cells.

## **Quattro Pro DOS Compatible Commands**

This command or menu equivalent is provided for compatibility with the DOS version of Quattro Pro. To learn more about it, see the Quattro Pro for DOS @Functions and Macros guide.

▶ **Related topics**





## Quattro Pro Macro Commands List

A  
B  
C  
D  
E  
▶  
G  
H  
I  
J  
K  
L  
M  
N  
O  
P  
Q  
R  
S  
T  
U  
V  
W  
X  
Y  
Z

## PerfectScript Macro Commands List

▶ \_\_\_\_\_ { }  
▶ \_\_\_\_\_ {;}  
▶ \_\_\_\_\_ ?

### A

▶ \_\_\_\_\_ ABS  
▶ \_\_\_\_\_ AbsoluteReference  
▶ \_\_\_\_\_ Activate  
▶ \_\_\_\_\_ Addmenu  
▶ \_\_\_\_\_ AddMenuItem  
▶ \_\_\_\_\_ AddSeries  
▶ \_\_\_\_\_ AddSubMenuItem  
▶ \_\_\_\_\_ ALT  
▶ \_\_\_\_\_ Alert  
▶ \_\_\_\_\_ Align.Option  
▶ \_\_\_\_\_ AnalysisExpert  
▶ \_\_\_\_\_ AnsiRead  
▶ \_\_\_\_\_ AnsiReadLn  
▶ \_\_\_\_\_ AnsiWrite  
▶ \_\_\_\_\_ AnsiWriteLn  
▶ \_\_\_\_\_ Anova1  
▶ \_\_\_\_\_ Anova2  
▶ \_\_\_\_\_ Anova3  
▶ \_\_\_\_\_ Application.Compatibility.Option

- ▶ Application.Country.Settings
- ▶ Application.Current.File
- ▶ Application.Display
- ▶ Application.Enable.Inspection
- ▶ Application.File.Options
- ▶ Application.General
- ▶ Application.International
- ▶ Application.Title
- ▶ Application.Property
- ▶ Assign
- ▶ AssignValue
- ▶ Audit.Remove.All.Arrows
- ▶ Audit.Trace.Dependents
- ▶ Audit.Trace.Precedents

## **B**

- ▶ BackSpace
- ▶ BackTab
- ▶ Beep
- ▶ BigLeft
- ▶ BigRight
- ▶ Blank
- ▶ BlockCopy
- ▶ BlockDelete.Option
- ▶ BlockFill.Option
- ▶ BlockInsert.Option
- ▶ BlockMove
- ▶ BlockMovePages
- ▶ BlockName.Option
- ▶ BlockReformat
- ▶ BlockTranspose
- ▶ BlockValues
- ▶ Branch
- ▶ Break
- ▶ BreakOff
- ▶ BreakOn
- ▶ BudgetExpert

## **C**

- ▶ Calc
- ▶ CapOff
- ▶ CapOn
- ▶ ChartExpert
- ▶ Choose
- ▶ Clear
- ▶ ClearComments
- ▶ ClearContents
- ▶ ClearFormats
- ▶ Close
- ▶ ColumnWidth
- ▶ Comment.Edit
- ▶ Comment.EditURL
- ▶ ComposeFormula
- ▶ Consolidate.Option
- ▶ ConsolidateExpert
- ▶ Contents
- ▶ Controls.Option
- ▶ CR
- ▶ CreateChart
- ▶ CreateObject
- ▶ CrossTab
- ▶ CrossTabReport.AddField
- ▶ CrossTabReport.CenterLabels
- ▶ CrossTabReport.ColumnSummary
- ▶ CrossTabReport.CopyStatic
- ▶ CrossTabReport.Create

- ▶ CrossTabReport.DataAlignment
- ▶ CrossTabReport.DefineFieldProps
- ▶ CrossTabReport.Destination
- ▶ CrossTabReport.DisplayInEmptyCell
- ▶ CrossTabReport.Edit
- ▶ CrossTabReport.EmptyCellString
- ▶ CrossTabReport.Expand
- ▶ CrossTabReport.FieldCmp
- ▶ CrossTabReport.FieldCmpBase
- ▶ CrossTabReport.FieldCmpltem
- ▶ CrossTabReport.FieldCmpltemPreset
- ▶ CrossTabReport.FieldHide
- ▶ CrossTabReport.FieldLabel
- ▶ CrossTabReport.FieldOptions
- ▶ CrossTabReport.FieldSummary
- ▶ CrossTabReport.FormatReport
- ▶ CrossTabReport.Hide
- ▶ CrossTabReport.LabelEdit
- ▶ CrossTabReport.MoveCell
- ▶ CrossTabReport.MoveField
- ▶ CrossTabReport.Name
- ▶ CrossTabReport.Options
- ▶ CrossTabReport.PageFilter
- ▶ CrossTabReport.PreserveDataFormat
- ▶ CrossTabReport.Refresh
- ▶ CrossTabReport.Remove
- ▶ CrossTabReport.RowSummary
- ▶ CrossTabReport.Show
- ▶ CrossTabReport.Source
- ▶ CrossTabReport.UpdateDataOnOpen
- ▶ CTRL

## **D**

- ▶ DatabaseQuery
- ▶ Date
- ▶ DbAlias
- ▶ Define
- ▶ Del
- ▶ Delete
- ▶ DeleteMenu
- ▶ DeleteMenuItem
- ▶ DelVar
- ▶ Descr
- ▶ DialogView
- ▶ DialogWindow.Property
- ▶ DialogWindow.Dimension
- ▶ DialogWindow.Disabled
- ▶ DialogWindow.Grid\_Options
- ▶ DialogWindow.Name
- ▶ DialogWindow.Position\_Adjust
- ▶ DialogWindow.Title
- ▶ DialogWindow.Value
- ▶ Dispatch
- ▶ DLL
- ▶ DLL.Load
- ▶ DoDialog
- ▶ Down
- ▶ DraftViewGoto
- ▶ Duplicate

## **E**

- ▶ Edit
- ▶ EditClear
- ▶ EditCopy
- ▶ EditCut

- ▶ EditGoto
- ▶ EditPaste
- ▶ End
- ▶ Esc
- ▶ Escape
- ▶ Eval
- ▶ Exec
- ▶ ExecAuto
- ▶ Execute
- ▶ ExecuteMacro
- ▶ Expon
- ▶ ExportGraphic

## **F**

- ▶ FileClose
- ▶ FileCloseAll
- ▶ FileCombine
- ▶ FileExit
- ▶ FileExtract
- ▶ FileImport
- ▶ FileNew
- ▶ FileOpen
- ▶ FileRetrieve
- ▶ FileSave
- ▶ FileSaveAll
- ▶ FileSaveAs
- ▶ FileSend
- ▶ FileSize
- ▶ FileVersion.Retrieve
- ▶ FileVersion.Retrieve\_Current
- ▶ FileVersionSave
- ▶ FloatCopy
- ▶ FloatCreate
- ▶ FloatMove
- ▶ FloatOrder.Option
- ▶ FloatSize
- ▶ FloatText
- ▶ Form
- ▶ For
- ▶ ForBreak
- ▶ Fourier
- ▶ Frequency.Option
- ▶ FTestv
- ▶ Functions

## **G**

- ▶ Get
- ▶ GetCellFormula
- ▶ GetCellValue
- ▶ GetDirectoryContents
- ▶ GetLabel
- ▶ GetNumber
- ▶ GetObjectPageContents
- ▶ GetObjectProperty
- ▶ GetProperty
- ▶ GetPos
- ▶ GetWindowList
- ▶ Goto
- ▶ Graph
- ▶ GraphChar
- ▶ GraphCopy
- ▶ GraphDeactivate
- ▶ GraphDelete
- ▶ GraphEdit
- ▶ GraphGallery

- ▶ GraphNew
- ▶ GraphSettings.Titles
- ▶ GraphSettings.Type
- ▶ GraphSettings.Check
- ▶ GraphSettings.Reset
- ▶ GraphView
- ▶ GraphWindow.Property
- ▶ Group.Option
- ▶ GroupObjects

## H

- ▶ Help
- ▶ HideErrorMessage
- ▶ HistoGram
- ▶ HLine
- ▶ Home
- ▶ HPage

## I

- ▶ If
- ▶ IfAutoObj
- ▶ IfKey
- ▶ ImFormat
- ▶ ImportGraphic
- ▶ ImportGraphic.ClipArt
- ▶ Indicate
- ▶ Initiate
- ▶ InitApp
- ▶ Ins
- ▶ Insert
- ▶ InsertBreak
- ▶ InsertObject
- ▶ InsertObject.DrawPicture
- ▶ InsertObject.TextArt
- ▶ InsertPageBreak.Option
- ▶ InsOff
- ▶ InsOn
- ▶ Inspect
- ▶ Invert.Option
- ▶ IsAutoObj

## J

No macros.

## K

No macros.

## L

- ▶ Left
- ▶ Let
- ▶ Link
- ▶ Links.Option
- ▶ Look

## M

- ▶ Macros

- ▶ MCorrel
- ▶ MCovar
- ▶ MapExpert
- ▶ MapNew
- ▶ Mark
- ▶ Menu
- ▶ MenuBranch
- ▶ MenuCall
- ▶ Message
- ▶ MoveAvg
- ▶ MoveTo
- ▶ MtGamt
- ▶ MtGrefi
- ▶ Multiply.Option

## **N**

- ▶ Name
- ▶ NamedStyle.Option
- ▶ Navigate.Option
- ▶ NextPane
- ▶ NextToPWin
- ▶ NextWin
- ▶ Notebook.Display
- ▶ Notebook.Property
- ▶ Notebook.Group\_Mode
- ▶ Notebook.Macro.Library
- ▶ Notebook.Password
- ▶ Notebook.Password.Level
- ▶ Notebook.Recalc.Settings
- ▶ Notebook.Summary.Option
- ▶ Notebook.System
- ▶ Notebook.Zoom\_Factor
- ▶ NumOff
- ▶ NumOn

## **O**

- ▶ ObjectsPageGoto
- ▶ OLE.Option
- ▶ OnError
- ▶ OnlineService
- ▶ Open
- ▶ Optimizer.Option
- ▶ Order.Option
- ▶ Outline.Option

## **P**

- ▶ Page.Property
- ▶ Page.Conditional\_Color
- ▶ Page.Default\_Width
- ▶ Page.Display
- ▶ Page.Name
- ▶ Page.Protection
- ▶ Page.TabColor
- ▶ Page.Zoom\_Factor
- ▶ PageViewGoto
- ▶ PanelOff
- ▶ PanelOn
- ▶ ParseExpert.ApplyFormatting
- ▶ ParseExpert.CellDelimiterString
- ▶ ParseExpert.CellDelimiterTypeComma
- ▶ ParseExpert.CellDelimiterTypeMultiSpace
- ▶ ParseExpert.CellDelimiterTypeOther
- ▶ ParseExpert.CellDelimiterTypeReturn

- ▶ ParseExpert.CellDelimiterTypeSpace
- ▶ ParseExpert.CellDelimiterTypeSemiColon
- ▶ ParseExpert.CellDelimiterTypeSpace
- ▶ ParseExpert.CellDelimiterTypeTab
- ▶ ParseExpert.ColumnWidths
- ▶ ParseExpert.ConsecutiveAsOne
- ▶ ParseExpert.DataType
- ▶ ParseExpert.DelimiterType
- ▶ ParseExpert.Go
- ▶ ParseExpert.IgnoreNonConformingRows
- ▶ ParseExpert.ConsecutiveAsOne
- ▶ ParseExpert.InputBlock
- ▶ ParseExpert.InputFile
- ▶ ParseExpert.InputType
- ▶ ParseExpert.JoinBrokenLines
- ▶ ParseExpert.LineLength
- ▶ ParseExpert.LoadSettings
- ▶ ParseExpert.LineLength
- ▶ ParseExpert.OtherDelimiter
- ▶ ParseExpert.OutputBlock
- ▶ ParseExpert.PageLength
- ▶ ParseExpert.PageLengthEnabled
- ▶ ParseExpert.Restore
- ▶ ParseExpert.RowDelimiterString
- ▶ ParseExpert.RowDelimiterTypeComma
- ▶ ParseExpert.RowDelimiterTypeMultiSpace
- ▶ ParseExpert.RowDelimiterTypeOther
- ▶ ParseExpert.RowDelimiterTypeReturn
- ▶ ParseExpert.RowDelimiterTypeSemiColon
- ▶ ParseExpert.RowDelimiterTypeSpace
- ▶ ParseExpert.RowDelimiterTypeTab
- ▶ ParseExpert.TextQualifier
- ▶ ParseExpert.ValueQualifier
- ▶ Parse.Option
- ▶ PasteFormat
- ▶ PasteLink
- ▶ PasteSpecial
- ▶ PauseMacro
- ▶ PGDN
- ▶ PlayPerfectScript
- ▶ POKE
- ▶ Preview
- ▶ Print.Option
- ▶ PTTestM
- ▶ PTTestV
- ▶ Put
- ▶ PutBlock
- ▶ PutBlock2
- ▶ PutCell
- ▶ PutCell2

## Q

- ▶ QGoto
- ▶ QUERY
- ▶ Query.Option
- ▶ QuickCorrect
- ▶ QuickFilter.Go
- ▶ QuickFilter.Toggle
- ▶ QuickFilter.TopGo
- ▶ QuickFunction
- ▶ Quit

## R

- ▶ Random
- ▶ RankPerc

- ▶ Read
- ▶ ReadLn
- ▶ Recalc
- ▶ RecalcCol
- ▶ RefreshMenuBar
- ▶ RefreshScreenOn
- ▶ Regress
- ▶ Regression.Option
- ▶ Request
- ▶ Resize
- ▶ ResizeToSame
- ▶ Restart
- ▶ RestrictInput.Option
- ▶ Return
- ▶ ReturnErrorValue
- ▶ Right
- ▶ RowColShow
- ▶ RowHeight

## **S**

- ▶ Sample
- ▶ SaveHtml!
- ▶ SaveHtml.BackgroundColor
- ▶ SaveHtml.FileOptions
- ▶ SaveHtml.GraphicType
- ▶ SaveHtml.Header
- ▶ SaveHtml.HeaderDescription
- ▶ SaveHtml.Layout
- ▶ SaveHtml.LineBeforeHeader
- ▶ SaveHtml.LineBeforeFooter
- ▶ SaveHtml.LinkColor
- ▶ SaveHtml.OutputFile
- ▶ SaveHtml.OutputType
- ▶ SaveHtml.SaveHtml
- ▶ SaveHtml.Source
- ▶ SaveHtml.TextColor
- ▶ SaveHtml.Title
- ▶ SaveHtml.UseBrowserColor
- ▶ SaveHtml.UserDetails
- ▶ SaveHtml.UseRGBValues
- ▶ SaveHtml.WallPaper
- ▶ Scenario.Option
- ▶ ScenarioExpert
- ▶ ScrollOff
- ▶ ScrollOn
- ▶ Search.Option
- ▶ SelectAll
- ▶ SelectBlock
- ▶ SelectFloat
- ▶ SelectObject
- ▶ Series.Option
- ▶ SeriesManager.Option
- ▶ SetCellString
- ▶ SetGraphAttr
- ▶ SetLCID
- ▶ SetMenuBar
- ▶ SetObjectProperty
- ▶ SetPos
- ▶ SetProperty
- ▶ Shift
- ▶ ShowErrorMessage
- ▶ Slide.Option
- ▶ SlideShowExpert
- ▶ SolveFor.Option
- ▶ Sort.Option
- ▶ SpeedFill
- ▶ SpeedFormat



- ▶ SpeedFormat.Option
- ▶ SpeedSum
- ▶ Step
- ▶ StepOff
- ▶ StepOn
- ▶ Subroutine
- ▶ SuppressErrorValue

## T

- ▶ Tab
- ▶ TabControl.Option
- ▶ Table
- ▶ TableLink.Option
- ▶ TableQuery.Option
- ▶ TableView
- ▶ TemplateTB.Option
- ▶ Terminate
- ▶ Toolbar
- ▶ TTestM

## U

- ▶ UIWorkspace.Exists
- ▶ UIWorkspace.GetCurrent
- ▶ UIWorkspace.Import
- ▶ UIWorkspace.Load
- ▶ UNDO
- ▶ UngroupObjects
- ▶ Up

## V

- ▶ VLine
- ▶ VPage

## W

- ▶ Wait
- ▶ WaitCursorOn
- ▶ WaitCursorOff
- ▶ WebQuery.Create
- ▶ WebQuery.Destination
- ▶ WebQuery.EditLink
- ▶ WebQuery.InitLink
- ▶ WebQuery.LinkRange
- ▶ WebQuery.LinkRefreshDuration
- ▶ WebQuery.LinkRefreshTime
- ▶ WebQuery.LinkRefreshType
- ▶ WebQuery.LinkWrapOption
- ▶ WebQuery.QueryFileName
- ▶ WebQuery.RefreshWebLink
- ▶ WebQuery.Run
- ▶ WebQuery.SetQueryOptions
- ▶ WebQuery.SetQueryParameters
- ▶ WebQuery.Source
- ▶ WhatIf.Option
- ▶ WhatIfExpert
- ▶ Window
- ▶ WindowArrlcon
- ▶ WindowCascade
- ▶ WindowClose
- ▶ WindowHide
- ▶ WindowMaximize
- ▶ WindowMinimize

- ▶ WindowMove
- ▶ WindowNewView
- ▶ WindowNext
- ▶ WindowPanes
- ▶ WindowOPW.Option
- ▶ WindowRestore
- ▶ WindowShow
- ▶ WindowSize
- ▶ WindowsOff
- ▶ WindowsOn
- ▶ WindowTile
- ▶ WindowTile.TileTopToBottom
- ▶ WindowTitles
- ▶ Workflow.RouteDocument
- ▶ Workflow.WorkFlowManager
- ▶ WorkSpace.Option
- ▶ Write
- ▶ WriteLn

## **X**

- ▶ XMLTag.AutoGenerate
- ▶ XMLTag.Create
- ▶ XMLTag.Delete
- ▶ XMLTag.Labels
- ▶ XMLTag.MakeTable
- ▶ XMLTag.Reset

## **Y**

No macros.

## **Z**

- ▶ Zoom
- ▶ ZTestM

## **Quattro Pro Macro Commands by Category List**

### **PerfectScript Macro Commands List**

- ▶ Keyboard Macro Commands
- ▶ Screen Macro Commands
- ▶ Interactive Macro Commands
- ▶ Program Flow Macro Commands
- ▶ Cell Macro Commands
- ▶ Command Equivalents
- ▶ OLE and DDE Macro Commands
- ▶ UI Building Macro Commands
- ▶ Object Macro Commands
- ▶ Analysis Tools Macro Commands
- ▶ Miscellaneous Macro Commands



**{ }**

### **Description**

{ } does nothing; use it to reserve space or insert blank lines in the macro without stopping it. Quattro Pro continues running the macro command following { }.

See [{ ; }](#) and [{STEPON}](#) for examples.

▶ **Related topics**

**{;}**

## Syntax

{;*String*}

## Description

{ ; } 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.

## Example

This example runs the `_int_update` subroutine, which calculates interest to date, then 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}
```

## Parameters

*String*      Numbers or letters up to 1020

► **Related topics**

## { ? }

### Description

{ ? } pauses macro execution and lets you press function keys, access menus, and choose commands until Enter is pressed. At that point the macro resumes execution.

Since this command gives you complete program control, use it with caution.

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 you 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.

### Example

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 you can enter a check number. The information you enter replaces the Check Number? prompt.

```
{EditGoto E15}
```

```
{PUTCELL "Check Number?"}
```

```
{?}~
```

### ► Related topics





## {ABS}

### Syntax

{ABS <Number>}

### Description

{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 F4 one time with the cell address selected; {ABS 2} is equivalent to pressing F4 twice, and so on, up to 8. If *Number* is missing, {ABS 1} is assumed.

{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 do not have to be edited if the absolute addresses are created before the formula is copied.

### Tips

- Include the {EDIT} command before {ABS} to make sure Edit mode is active.
- When you convert Quattro Pro macros to PerfectScript macros, the following Quattro Pro macro commands must be replaced by the corresponding PerfectScript commands:

Quattro Pro	PerfectScript
{ABS}	AbsoluteReference()
{ASSIGN}	AssignValue()

### Example

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)}
```

### Parameters

*Number* 1 through 8; 1 = pressing F4 once, 2 = pressing F4 twice, and so on (optional)

### ▶ Related topics

# {AbsoluteReference}

## Syntax

{AbsoluteReference <Number>}

## PerfectScript Syntax

AbsoluteReference([Number: Numeric])

## Description

Lets you convert relative cell addresses to absolute addresses. *Number* provides control over what part of the formula converts to an absolute address.

## Tip

- When you convert Quattro Pro macros to PerfectScript macros, the following Quattro Pro macro commands must be replaced by the corresponding PerfectScript commands:

<b>Quattro Pro</b>	<b>PerfectScript</b>
{ABS}	AbsoluteReference()
{ASSIGN}	AssignValue()

## Parameters

*Number*            1 through 8; 1 = pressing F4 once, 2 = pressing F4 twice, and so on (optional)

## ► Related topics

## {ACTIVATE}

### Syntax

{ACTIVATE *WindowName*}

### PerfectScript Syntax

Activate (*WindowName*:String)

### Description

{ACTIVATE} makes the window specified by the string *WindowName* active. You can find the name of a window on its title bar.

#### ► Note

The *WindowName* parameter must be exactly the same as what appears in the title bar. For example, if the title bar displays "C:\My Documents\Notebk1.QPW (unmodified)", then the ACTIVATE command must read {ACTIVATE "C:\My Documents\Notebk1.QPW (unmodified)"}

### Example

To make the named chart PROFITS (in the notebook REPORT.WB3) active, use

```
{ACTIVATE "C:\SALES\REPORT.WB3:PROFITS" }
```

Use the same syntax for activating dialog windows. To make the notebook itself active, use

```
{ACTIVATE "C:\SALES\REPORT.WB3" }
```

### Parameters

*WindowName*      Name of the window to make active  
e

## {ADDMENU}

### Syntax

{ADDMENU *MenuPath*, *MenuBlk*}

### PerfectScript Syntax

AddMenu (*MenuPath*:String; *MenuBlock*:String)

### Description

{ADDMENU} lets you add menus to the active menu system. (Use [{ADDMENUITEM}](#) to add individual menu items 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 Edit menu, use /Edit; to insert a menu before the Copy command on the Edit menu, use /Edit/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 first item on the File menu.

You can also use numbers to identify menu items. 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 Properties).

*MenuBlk* includes the cells containing a menu definition. *MenuBlk* must include all cells in the new menu.

### Tips

- You can add new menus only to the menu bar on either side of the Edit and Tools menus--that is, one position to the left or right of the Edit menu and one position to the left or right of the Tools menu. The area between these menu positions is reserved for menus that change depending on the active window. Likewise, you cannot delete menus within this menu, either. You can add menu items to menus between the Edit and Tools menus, but the new menu items will be swapped out of the menu when the context changes.
- Changes made to the menu system using this command are not saved; they are lost when you exit Quattro Pro. Each time you run a macro containing {ADDMENU}, the menu changes appear again.
- To restore the original menu bar, use the macro command [{SETMENUBAR}](#) without an argument.

### Parameters

*MenuPath*      Location in the menu system to insert a new menu  
*MenuBlk*      Location in the menu system to insert a new menu

#### ► Note

- This command is obsolete.

▶ **Related topics**

# {ADDMENUITEM}

## Syntax

{ADDMENUITEM *MenuPath*, *Name*, <*Link*>, <*Hint*>, <*HotKey*>, <*DependString*>, <*Checked*>}

## PerfectScript Syntax

AddMenuItem (MenuPath:String; ItemName:String; [Link:String]; [Hint:String]; [HotKey:String]; [DependString:String]; [Checked?:Enumeration {Yes!; No!}])

## Description

{ADDMENUITEM} is like [{ADDMENU}](#), but inserts a single menu item before *MenuPath* instead of a new menu.

See the description of {ADDMENU} for the syntax of *MenuPath*. *Name* is the name of the new menu item; if it is a command, precede its underlined letter with an ampersand (&).

*Link* specifies the actions the menu item performs (for example, "MACRO \_remove\_file" runs the macro \_remove\_file).

## Example

The following macro adds the menu item Find Object above Edit ► Go To. Find Object runs a macro command called \_FINDOBJ.

```
{ADDMENUITEM "/Edit/Go To","Find Object", "MACRO _FINDOBJ", "Finds a floating object on the notebook sheet",  
"Ctrl+Shift+F", "No, Yes, No, No, No, No", "No"}
```

## Parameters

<i>MenuPath</i>	Location in the menu system to insert a new menu item; enter the sequence of menu items separated by forward slashes (/); you can use <- and -> to place a menu at the top or before the bottom of a menu, respectively. For example, /File/<- specifies the first item on the File menu. You can also use numbers to identify menu items. For example, /File/0 specifies the first item on the File menu (the ID numbers start at zero).
<i>Name</i>	Name of the command to add; if you want a letter of the name to appear underlined, precede it with an ampersand (&); to add a divider line, type a series of hyphens (-).
<i>Link</i>	Action to perform when the command is chosen (optional); this argument can specify a link command or a macro command to run when the menu item is chosen; click here <a href="#">»»</a> for details.
<i>Hint</i>	Help text to display in a pop-up window when the command is highlighted (optional)
<i>HotKey</i>	Shortcut key that chooses the command (optional); separate key combinations with a plus sign (+), for example, Alt+F4.
<i>DependString</i>	Areas in which the command is available (optional); enter Yes or No for each area, separated by commas, in the following order: desktop, notebook, chart Window, dialog window, input line, Objects sheet. Example: "No, No, Yes, No, No, No" makes the menu item available only when the chart window is active.
<i>Checked</i>	Type "Yes" if the command should have a checkmark display by it (optional)

## Tips

- You can add menu items to any menu, but if you change a context-sensitive menu (all menus between Edit and Tools on the menu bar), the change applies only to the menu in the active window. For example, suppose you use a macro to change the View menu when the notebook window is active. If you then open a chart window, the chart View menu appears--without the change. If you want the change to apply to that View menu as well, you must run the macro again.
- Changes made to the menu system using this command are not saved; they are lost when you exit Quattro Pro. Each time you run a macro containing {ADDMENUITEM}, the menu changes appear again.
- To restore the original menu bar, use the macro command [{SETMENUBAR}](#) without an argument.

### ► Note

- This command is obsolete.

### ► [Related topics](#)

## **{ADDSERIES}**

### **Syntax**

{ADDSERIES *Block*, *Name*}

### **PerfectScript Syntax**

AddSeries (Block:String; Name:String)

### **Description**

{ADDSERIES} adds a data series to a floating chart. Use {ADDSERIES} as an equivalent to dragging cells onto a chart on a notebook sheet to add a series.

### **Example**

The following macro adds the series contained in the cells A:E3..E13 to the floating chart named BUDGET:

```
{ADDSERIES A:E3..E13, BUDGET}
```

### **Parameters**

*Block*        Cells containing a data series  
*Name*        Name of the chart to which you want to add a series

### **▶ Related topics**

## {ADDSUBMENUITEM}

### Syntax

{ADDSUBMENUITEM *MenuPath*, *Name*, <*Link*>, <*Hint*>, <*HotKey*>, <*DependString*>, <*Checked*>}

### PerfectScript Syntax

AddSubMenuItem (MenuPath:String; ItemName:String; [Link:String]; [Hint:String]; [HotKey:String]; [DependString:String]; [Checked?:Enumeration {Yes!; No!}])

### Description

{ADDSUBMENUITEM} is like {[ADDMENUITEM](#)}, but converts the command indicated by *MenuPath* into a submenu and adds the new command to the submenu.

*Link* specifies the actions the submenu item performs (for example, "MACRO \_remove\_file" runs the macro \_remove\_file).

### Parameters

<i>MenuPath</i>	Location in the menu system to insert a new menu item; enter the sequence of menu items separated by forward slashes (/); you can use <- and -> to place a menu at the top or before the bottom of a menu, respectively. For example, /File/<- specifies the first item on the File menu. You can also use numbers to identify menu items. For example, /File/0 specifies the first item on the File menu (the ID numbers start at zero).
<i>Name</i>	Name of the command to add; if you want a letter of the name to appear underlined, precede it with an ampersand (&)
<i>Link</i>	Action to perform when the command is chosen (optional); this argument can specify a link command or a macro command to run when the menu item is chosen; click here <a href="#">»</a> for details.
<i>Hint</i>	Help text to display on the status line when the command is highlighted (optional)
<i>HotKey</i>	Shortcut key that chooses the command (optional); separate key combinations with a plus sign (+), for example, Alt+F4.
<i>DependString</i>	Areas in which the command is available (optional); enter Yes or No for each area, separated by commas, in the following order: desktop, notebook, chart window, dialog window, input line, Objects sheet. Example: "No, No, Yes, No, No, No" makes the menu item available only when the chart window is active.
<i>Checked</i>	Type "Yes" if the command should have a checkmark display by it (optional)

### Tips

- You can add menu items to any menu, but if you change a context-sensitive menu (all menus between Edit and Tools on the menu bar), the change applies only to the menu in the active window. For example, suppose you use a macro to change the View menu when the notebook window is active. If you then open a chart window, the chart View menu appears--without the change. If you want the change to apply to that View menu as well, you must run the macro again.
- Changes made to the menu system using this command are not saved; they are lost when you exit Quattro Pro. Each time you run a macro containing {ADDSUBMENUITEM}, the menu changes appear again.

#### ► Note

- This command is obsolete.

#### ► Related topics

## **{Alert}**

### **Syntax**

{Alert *Title*; *Message*; *OKExit?*<; *Type*; *Icon*; *DefaultBtn*>}

### **PerfectScript Syntax**

Alert (*Title?*:String; *Message?*:String; *OKExit?*:String; [*Type?*:Numeric]; [*Icon?*:Numeric]; [*DefaultBtn?*:Numeric])

### **Description**

*Message* displays a dialog box "message" for the user of the macro to manipulate. You set the title and message text of the dialog via the *Title* and *Message* arguments.

*OKExit* stores the result of the dialog box, so the macro can determine which button was pushed to close it.

*Type* specifies what type of message box will appear. It can be a message box with just an OK button, or one with both an OK and Cancel, etc.

*Icon* specifies which graphic to use on the above dialog. Zero represents the Error icon you would see on a regular error message under Windows. One is for the Question Mark icon, etc.

*DefaultBtn* determines which button (if there are multiple) is the "default" button. If this argument is a number greater than the number of buttons on the dialog, then the first button will be default.

### **Parameters**

<i>Title</i>	Title of resultant dialog.
<i>Message</i>	Message text of resultant dialog.
<i>OKExit?</i>	Cell to store how the dialog box closed (1 for OK, 2 for Cancel, 3 for Abort, 4 for Retry, 5 for Ignore, 6 for Yes, 7 for No).
<i>Type</i>	0 for dialog with just an OK button, 1 for OK/Cancel, 2 for Abort/Retry/Ignore, 3 for Yes/No/Cancel, 4 for Yes/No, and 5 for Retry/Cancel (optional; 0 is default).
<i>Icon</i>	0 for icon of type Error, 1 for Question, 2 for Warning, 3 for Info (optional; 0 is the default).
<i>DefaultBtn</i>	0 for first button being default, 1 for second, etc... (optional; 0 is the default).



## {Align}

### Description

Aligns objects in the Chart window and the Dialog window. Its action depends on whether a chart or dialog window is active.

Value is the amount of space to leave between controls, in pixels.

### Options

{Align.Bottom}	Lines up the bottom edges of the selected objects.
{Align.Horizontal_Center}	Lines up the centers of the objects as though they were beads on an invisible horizontal string
{Align.Horizontal_Space Value}	Horizontally spaces objects by the amount of pixels you specify
{Align.Left}	Lines up the left sides of the selected objects
{Align.Right}	Lines up the right sides of the selected objects
{Align.Top}	Lines up the top edges of the selected objects
{Align.Vertical_Center}	Lines up the centers of the objects as though they were beads on an invisible vertical string
{Align.Vertical_Space Value}	Vertically spaces objects by the amount of pixels you specify

## {ALT}

### Syntax

{ALT+Key <Number>}

### Description

{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} and {SHIFT}. For example, {CTRL+SHIFT+D} is equivalent to pressing Ctrl+Shift+D, the Date keys.

### Example

{ALT+F} emulates pressing Alt+F, which could activate a control in a dialog box with an underlined letter of F.

### Parameters

*Key* Keyboard macro command (PGUP, DOWN, and so on)

*Numbe* Number of times to repeat the operation (optional)

*r*

#### ► Note

- This command is obsolete.

#### ► Related topics

## **{AnalysisExpert}**

### **Description**

{AnalysisExpert} performs a number of advanced statistical, numerical, and financial analysis tasks. The macro has no arguments. {AnalysisExpert} displays the first Analysis Tools Expert dialog box.

Before you use an analysis tool, make sure the input cells you are analyzing are arranged properly and contain the right kind of data (that is, numeric data, not strings). The analysis tools have varying restrictions on the contents of the input cells and size of the cell area.

## **{ANOVA1}**

### **Syntax**

{ANOVA1 *InBlock*, *OutBlock*, <*Grouped*>, <*Labels*(0|1)>, <*Alpha*>}

### **PerfectScript Syntax**

ANOVA1 (*InBlock*:String; *OutBlock*:String; [*Grouped*:String]; [*Labels?*:Enumeration {Yes!; No!}]; [*Alpha*:Numeric])

### **Description**

{ANOVA1} performs a one-way analysis of variance. Use {ANOVA1} to test whether two or more samples come from the same population. {ANOVA1} is equivalent to the Anova: One-Way analysis tool.

### **Parameters**

<i>InBlock</i>	Input cells containing two or more sets of numeric data arranged in columns or rows
<i>OutBlock</i>	Upper left cell of the output cells
<i>Grouped</i>	"C" to group results by column or "R" to group results by row; the default is "C"
<i>Labels</i>	1 if labels are located in the first column or row of the input cells; 0 if the input cells do not contain labels; the default is 0
<i>Alpha</i>	The significance level at which to evaluate values for the F-statistic; the default is 0.05

### **► Related topics**

## **{ANOVA2}**

### **Syntax**

{ANOVA2 *InBlock*, *OutBlock*, *SampleRows*, <*Alpha*>}

### **PerfectScript Syntax**

ANOVA2 (*InBlock*:String; *OutBlock*:String; *SampleRows*:Numeric; [*Alpha*:Numeric])

### **Description**

{ANOVA2} performs a two-way analysis of variance, with more than one sample for each group of data. {ANOVA2} is equivalent to the Anova: Two-Way with Replication analysis tool.

### **Parameters**

<i>InBlock</i>	Input cells containing two or more sets of numeric data arranged in columns; the first row must contain labels for each group; the first column must contain row labels indicating the beginning of each sample
<i>OutBlock</i>	Upper-left cell of the output cells
<i>SampleRows</i>	The number of rows in each sample
<i>s</i>	
<i>Alpha</i>	The significance level at which to evaluate values for the F-statistic; the default is 0.05

### **► Related topics**

## **{ANOVA3}**

### **Syntax**

{ANOVA3 *InBlock*, *OutBlock*, <*Labels*(0|1)>, <*Alpha*>}

### **PerfectScript Syntax**

ANOVA3 (*InBlock*:String; *OutBlock*:String; [*Labels?*:Enumeration {Yes!; No!}]; [*Alpha*:Numeric])

### **Description**

{ANOVA3} performs a two-way analysis of variance, with only one sample for each group of data. {ANOVA3} is equivalent to the Anova: Two-Way Without Replication analysis tool.

### **Parameters**

<i>InBlock</i>	Input cells containing two or more sets of numeric data arranged in columns or rows
<i>OutBlock</i>	Upper-left cell of the output cells
<i>Labels</i>	1 if labels are located in the first column or row of the input cells; 0 if the input cells do not contain labels; the default is 0
<i>Alpha</i>	The significance level at which to evaluate the F-statistic; the default is 0.05

### **► Related topics**

## **{ANSIREAD}**

### **Syntax**

{ANSIREAD *#Bytes*, *Location*}

### **Description**

{ANSIREAD} 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*, like {READ} but without any character mapping. This macro is provided for international users. See [{READ}](#), for more information.

### **Parameters**

*#Bytes*        Number of bytes of characters to read from a file  
*Location*     Cell in which to store the characters read

► [Related topics](#)

## **{ANSIREADLN}**

### **Syntax**

{ANSIREADLN *Location*}

### **Description**

{ANSIREADLN} is like [{ANSIREAD}](#), but instead of using a number of bytes to determine the amount of text to read, {ANSIREADLN} reads forward from the current file pointer location up to and including the carriage-return/linefeed at the end of the line, like {READLN} but without any character mapping. This macro is provided for international users. See [{READLN}](#) for more information.

### **Parameters**

*Location*      Cell in which to store the characters read

► [Related topics](#)

## **{ANSIWRITE}**

### **Syntax**

{ANSIWRITE *String*,<*String2*>,<*String3*,...>}

### **Description**

{ANSIWRITE} copies *String(s)* to a file opened with the [{OPEN}](#) command, starting at the location of the file pointer, like {WRITE} but without any character mapping. This macro is provided for international users. See [{WRITE}](#), for more information.

### **Parameters**

*String* String of characters to be written into the open file

### **▶ [Related topics](#)**



## **{ANSIWRITELN}**

### **Syntax**

{ANSIWRITELN *String*,<*String2*>,<*String3*,...>}

### **Description**

{ANSIWRITELN} 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, like {WRITELN} but without any character mapping. This macro is provided for international users. See {WRITELN}, for more information.

### **Parameters**

*String* String of characters to be written into the open file as a single line

### **▶ Related topics**

## **{Application}**

### **Syntax**

{Application.*Property*}

### **Description**

{Application} changes application properties such as compatibility options, display options, international options, macro and menu options, file options, and general options. Some settings appear only in Developer mode.

You can use {Application?} or {Application!} to display the Application dialog box. {Application?} lets the user manipulate the dialog box, whereas {Application!} relies on the macro to manipulate it.

## {Application.Compatibility.Option}

### Syntax

{Application.Compatibility<.Option>}

### PerfectScript Syntax

Application\_Compatibility(<.Option>)

### Description

Equivalent to Tools ► Settings  
► Compatibility

### Parameters

<i>AlternateMenuBar</i> [String]	Lets you specify which menu to use. "Quattro Pro 8/9" "Quattro Pro 7" "Excel 97" "Custom"
<i>AutoArrayWrap</i> [Boolean]	Lets you specify whether CTRL+SHIFT+ENTER generates an @ARRAY function, or whether Quattro Pro automatically determines whether one is needed. 0 CTRL+SHIFT+ENTER generates an @ARRAY function 1 Quattro Pro automatically determines whether one is needed
<i>CompatibilityMode</i> [String]	Lets you specify which compatibility default is used. "Quattro Pro 9" "Quattro Pro 8" "Excel 97" "Custom"
<i>Def_Columns_Limit</i> [Numeric]	Lets you specify the maximum number of columns a notebook can contain.
<i>Def_Rows_Limit</i> [Numeric]	Lets you specify the maximum number of rows a notebook can contain.
<i>Def_Sheets_Limit</i> [Numeric]	Lets you specify the maximum number of sheets a notebook can contain.
<i>File_Extension</i> [String]	Lets you specify the default file format.
<i>Min_Number_Sheets</i> [Numeric]	Lets you specify the minimum number of sheets a notebook can contain.
<i>Range_Syntax</i> [String]	Equivalent to Tools ► Settings ► Compatibility ► 3D Syntax.
<i>Sheet_Tab_Label</i> [Boolean]	Equivalent to Tools ► Settings ► Sheet Tab Display ► Display as Numbers. This option is obsolete.

### ► Related topics

## **{Application.Country\_Settings}**

### **Syntax**

```
{Application.Country_Settings "Symbol, Prefix|Suffix, Country"}
```

### **PerfectScript Syntax**

```
Application_Country_Settings (Settings:String)
```

### **Description**

{Application.Country\_Settings} sets the type of currency symbol and its placement before or after values for a particular country.

This macro replaces previous Quattro Pro macros, {Application.International.Currency\_Symbol} and {Application.International.Placement}.

### **Example**

The following macro sets the currency symbol to \$ and places the symbol before values for United States currency values.

```
{Application.Country_Settings "$,Prefix,United States"}
```

### **▶ Related topics**

## **{Application.Current\_File}**

### **Description**

{Application.Current\_File} returns the name of the active notebook. This command equivalent is used only with @COMMAND.

▶ **Related topics**

# {Application.Display}

## Syntax

{Application.Display<Option>}

## PerfectScript Syntax

Application\_Display (Settings:String)

## Description

{Application.Display} lets you specify cell 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.

## Example

The following macro command hides the time, hides the standard Toolbar, displays the input line and status line, sets the cell syntax to standard, hides the Property Bar, and displays the scroll indicators and QuickTips.

```
{Application.Display "None,No,Yes,Yes,A..B:A1..B2,No,Yes,Yes"}
```

## Options

{Application.Display "Toolbar, InputLine, Status, RangeSyntax, PropBand, ScrollIndicator, Hint, DefaultView, SheetTabLabel, MinNumSheets, ShowGroupboxAsLine, ShowPreselection, ShowHistoryList"}

{Application.Display.Clock\_Display Yes|No}

{Application.Display.CommentMarkers Yes|No}

{Application.Display.Default\_View Draft|Page}

{Application.Display.Default\_Zoom Yes|No}

{Application.Display.FormulaMarkers Yes|No}

{Application.Display.History\_List Yes|No}

{Application.Display.Min\_Number\_Sheets N}

{Application.Display.Range\_Syntax "A..B:A1..B2"|"A:A1..B:B2"}

{Application.Display.RealTime\_Prev Yes|No}

{Application.Display.Sheet\_Tab\_Label Letters|Numbers}

{Application.Display.Sheet\_Tab\_Label Letters|Numbers}

{Application.Display.Shortcut\_Keys Yes|No}

Lets you specify whether to show or hide portions of the Quattro Pro window, and switches between 3-D syntax schemes.

Lets you specify whether to show the Clock Display.

**This option is obsolete.**

0 Do not show the Clock Display.

1 Show the Clock Display.

Lets you specify whether to show the Comment Markers

0 Do not show the Comment Markers.

1 Show the Comment Markers.

Lets you specify whether new Notebooks come up in Draft view or Page Preview view. **This option is obsolete.**

Lets you specify whether to enable the Default Zoom. **This option is obsolete.**

0 Do not enable the Default Zoom.

1 Enable the Default Zoom.

Lets you specify whether to display the Formula Markers.

0 Do not display the Formula Markers.

1 Display the Formula Markers.

Lets you specify whether to display the File History List off the File menu.

0 Do not display the File History List.

1 Display the File History List.

Lets you specify the default number of sheets on new Notebooks. **This option is obsolete.**

Lets you switch between 3-D syntax schemes. **This option is obsolete.**

Lets you specify whether to enable the RealTime Preview.

0 Do not enable the RealTime Preview.

1 Enable the RealTime Preview.

Allows you to choose whether your default sheet tab names are letters (A..IV) or numeric (Sheet1..Sheet256). **This option is obsolete.**

Toggles dialog Group Boxes between being 'boxes' or just a line above the group. **This option is obsolete.**

Lets you specify whether to display shortcut keys.

{Application.Display.Show\_GroupBox\_As\_Line Yes|No}

{Application.Display.Show\_InputLine Yes|No}

{Application.Display.Show\_PreSelection Yes|No}

{Application.Display.Show\_Property\_Band Yes|No}

{Application.Display.Show\_Scroll\_Indicator Yes|No}

{Application.Display.Show\_StatusLine Yes|No}

{Application.Display.Show\_Tool\_Hint Yes|No}

{Application.Display.Show\_Toolbar Yes|No}

#### ► Related topics

0 Display shortcut keys.

1 Do not display shortcut keys.

Toggles dialog Group Boxes between being a 'box', or just a line above the group. **This option is obsolete.**

Lets you specify whether to show the Input Line.

0 Do not show the Input Line

1 Show the Input Line.

Toggles Windows buttons and other controls between being 3-D and flat. **This option is obsolete.**

Lets you specify whether to show the Property Bar. **This option is obsolete.**

0 Do not show the Property Bar

1 Show the Property Bar

Lets you specify whether to show the Scroll Indicators.

0 Do not show the Scroll Indicators.

1 Show the Scroll Indicators.

Lets you specify whether to show the Application Bar. **This option is obsolete.**

0 Do not show the Application Bar.

1 Show the Application Bar.

Lets you specify whether to show QuickTips.

0 Do not show QuickTips

1 Show QuickTips

Lets you specify whether to show the toolbar. **This option is obsolete.**

0 Do not show the toolbar

1 Show the toolbar

## **{Application.Enable\_Inspection}**

### **Syntax**

{Application.Enable\_Inspection Yes|No}

### **PerfectScript Syntax**

Application\_Enable\_Inspection (Enable?:Enumeration {Yes!; No!})

### **Description**

{Application.Enable\_Inspection} enables (Yes) or disables (No) Object Inspector menus. It is available only in Developer mode.

### **▶ Related topics**



## {Application.File\_Options}

### Syntax

{Application.File\_Options<Option>}

### PerfectScript Syntax

Application\_File\_Options (Settings:String)

### Description

{Application.File\_Options} includes information that is used every time you start Quattro Pro. It lets you specify the startup folder, autoload file, default file extension, and other options. The arguments of {Application.File\_Options} (which sets all options of the File Options property in one command) use the same syntax as those in the {Application.File\_Options.Option} commands.

{Application.File\_Options.AutoBack\_Enabled} and {Application.File\_Options.AutoBack\_Time} enable the creation of temporary backup files at a specified time interval.

{Application.File\_Options.Autoload\_File} sets the file to be loaded every time Quattro Pro is started.

{Application.File\_Options.File\_Extension} sets the default file extension to be used with file-handling commands.

{Application.File\_Options.Full\_Path\_Titles} shows the full path of notebook files in the title bar of the notebook window.

{Application.File\_Options.QuickTemplates} enables or disables the use of notebook templates when you create a new notebook.

{Application.File\_Options.Startup\_Directory} sets the directory initially displayed by file-handling commands.

{Application.File\_Options.TempDir} specifies the directory containing QuickTemplate files.

### Example

The following macro command sets the startup directory to C:\COREL\SUITE8, sets the autoload file to QUATTRO.WB3, sets the file extension to .WB3, enables autobackup at 15-minute intervals, enables the display of full path titles, enables QuickTemplates, sets the QuickTemplate directory, and sets the custom @function directory.

```
{Application.File_Options "C:\COREL\SUITE8\, QUATTRO.WB3, WB3, Yes, 15,
Yes,, Yes,
0 C:\COREL\SUITE8\TEMPLATE, C:\COREL\SUITE8"}
```

### Options

{Application.File_Options StartupDir, AutoFile, FileExt, AutoBackup?(Yes No),AutoBackupTime, FullPathTitles?(Yes No), AutoBack,QuickTemplates?(Yes No), QuickTemplateDir, URLUpdateTime, UpdateURL}	Open File Options dialog box.
{Application.File_Options.AutoBack_Enabled Yes No}	Create timed backup files at specified intervals.
{Application.File_Options.AutoBack_Time Integer}	Set the amount of time between automatic backups.
{Application.File_Options.Autoload_File String}	Open a file automatically when you start Quattro Pro.
{Application.File_Options.AutoRefreshTime N}	Specify how many minutes should pass before URLs refresh. This option is obsolete.
{Application.File_Options.DoRefresh Yes No}	Refresh URLs at specified time intervals. This option is obsolete.
{Application.File_Options.File_Extension String}	Specify a default file extension. This option is obsolete.
{Application.File_Options.Full_Path_Titles Yes No}	Show full folder paths in title bars.
{Application.File_Options.QuickTemplates Yes No}	Enable QuickTemplates. <b>This option is obsolete.</b>
{Application.File_Options.Startup_Directory String}	Specify a default folder.
{Application.File_Options.TempDir Path}	Specify a folder for QuickTemplates. <b>This option is obsolete.</b>
{Application.File_Options.WPDialogs Yes	Use enhanced file dialogs.

No}

▶ **Related topics**

## {Application.General}

### Syntax

{Application.General<Option>}

### PerfectScript Syntax

Application\_General (Settings:String)

### Description

{Application.General} lets you:

- enable the Edit ► Undo command
- make a variety of keys work in the same way as in Quattro Pro for DOS
- set the behavior of the cell selector when you press Enter
- specify how long to wait before changing from cell selection to Drag-and-Drop mode
- specify whether to use formula entry from Quattro Pro version 5

### Example

The following macro command enables Undo, makes the cell selector move down when you enter data, sets the cell drag and drop delay time to 400 milliseconds, and uses Quattro Pro formula entry.

```
{Application.General "Yes,No,Yes,Yes,400,No,No,No,No" }
```

### Options

{Application.General "UseUndo?(Yes No), CompatibleKeys?(Yes No), MoveCellOnEnter?(Yes No),, DelayTime, Compatible_Formula_Entry?(Yes No), Fit-As-You-Go?(Yes No), Calc-As-You-Go?(Yes No),QuickType?(Yes No), CellReferenceChecker?(Yes No)}	Opens the General Options tab.
{Application.General.Calc-As-You-Go Yes No}	Turns on/off Calc As You Go.
{Application.General.Cell_Reference_Checker Yes No}	Turns on/off the Cell Reference Checker.
{Application.General.Compatible_Formula_Entry Yes No}	Sets how you want to enter formulas.
{Application.General.Compatible_Keys Yes No}	Makes a variety of keys work the same way as in Quattro Pro for DOS.
{Application.General.Delay_Time Integer}	Specifies how long to wait before changing from cell selection to Drag and Drop mode.
{Application.General.Direction Down Up Left Right}	Equivalent to Tools ► Settings ► General ► Direction.
{Application.General.Fit-As-You-Go Yes No}	Turns on/off Fit-As-You-Go, which automatically sizes columns on data entry.
{Application.General.MoveCellOnEnterKey Yes No}	Makes the selector move down a cell every time you enter data.
{Application.General.QuickType Yes No}	Turns on/off QuickType, which as you type a label or function, finds the closest match.
{Application.General.Undo Yes No}	Enables the Undo feature.

### ► Related topics

# {Application.International}

## Syntax

{Application.International<*Option*>}

## PerfectScript Syntax

Application\_International (Settings:String)

## Description

{Application.International} 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 *Negative* argument can be Signed or Parens, the same settings that {Application.International.Negative} accepts.

To set the currency symbol and its placement either before or after values, use the [{Application.Country\\_Settings}](#) macro.

## Example

The following macro command specifies that the Quattro Pro currency format is used with parentheses to indicate negative values, sets the punctuation, sets the date and time formats to Windows defaults, sets the sort order to English, disables LICS conversion, and sets the country used for currency to United States. The entire string must be enclosed within a set of quotes. (Enter all of the example into one cell.)

```
{Application.International ", Quattro Pro,, Parens,""1,234.56 (a1,a2)"",  
Windows Default, Windows Default, Quattro Pro, English (American), No,  
United States"}
```

## Options

{Application.International ", Currency, , Negative, Punctuation, DateFmt, TimeFmt, Language, Conversion, Country"}	Opens the International tab
{Application.International.Currency "Windows Default" "Quattro Pro"}	Sets the default currency symbol
{Application.International.Date_Format <i>String</i> }	Determines the international formats given as options for date display
{Application.International.Language <i>String</i> }	Equivalent to Tools ► Settings ► International ► LanguageMode
{Application.International.Language <i>String</i> }	Selects an interface language
{Application.International.LanguageMode SuiteDefault  Quattro Pro}	Equivalent to Tools ► Settings ► International ► Language.
{Application.International.LICS_Conversion Yes No}	Converts Lotus International Character Set characters into standard ANSI characters
{Application.International.Negative Signed  Parens}	Controls whether negative values are preceded by a minus sign or surrounded by parentheses. This option is obsolete.
{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"}	Controls the characters used as thousands, decimal, and argument separators
{Application.International.Time_Format <i>String</i> }	Determines the international formats given as options for time display

## ► [Related topics](#)

## {Application.Macro}

### Syntax

{Application.Macro<Option>}

### PerfectScript Syntax

Application\_Macro (Settings:String)

### Description

{Application.Macro.Option} lets you control screen updates, display alternative menu systems, and run startup macros when you open a notebook. 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.

### Example

The following macro command specifies that windows should not display when a macro runs, makes the slash key display the Quattro Pro for DOS menu system, and sets the startup macro to BUDGET (Quattro Pro will run a macro named BUDGET whenever a notebook is opened containing a macro by that name).

```
{Application.Macro "Window,,Quattro Pro - DOS,BUDGET"}
```

### Options

{Application.Macro "MacSuppress,, SlashKey, StartupMacro"}	Opens the Macro tab
{Application.Macro.Macro_Redraw Both None Panel Window}	Suppresses redrawing of the window, panels, or both
{Application.Macro.Slash_Key MenuName}	Controls which menu system displays when you press the slash key. <b>This option is obsolete.</b>
{Application.Macro.Startup_Macro String}	Sets the macro to run every time you open a notebook containing a macro with this name

### ► Related topics

## **{Application.Title}**

### **Syntax**

{Application.Title *Title*}

### **PerfectScript Syntax**

Application\_Title (Title:String)

### **Description**

{Application.Title *Title*} changes the title displayed on Quattro Pro's title bar. This property is available only after starting Quattro Pro in developer mode (with /D parameter).

### **▶ Related topics**

## **{ASSIGN}**

### **Syntax**

{ASSIGN *VarExpr*, *ValExpr*}

### **Description**

The {ASSIGN} macro command is equivalent to the assignment statement *variable=value* in a programming language.

### **Tip**

- When you convert Quattro Pro macros to PerfectScript macros, the following Quattro Pro macro commands must be replaced by the corresponding PerfectScript commands:

<b>Quattro Pro</b>	<b>PerfectScript</b>
{ABS}	AbsoluteReference()
{ASSIGN}	AssignValue()

### **Example**

{ASSIGN calc, CreateObject("DispCalc.Application")} creates an object of the DispCalc application and assigns it to a named variable calc.

{ASSIGN calc.accum, 0} clears the accumulated value of DispCalc.

{ASSIGN calc.accum, @SUM(A1..A10)} assigns the sum of A1..A10 to the accumulated value of DispCalc.

For more details on using {ASSIGN} and other OLE automation macro commands, see [Using OLE Automation Features](#).

### **Parameters**

*VarExpr*      A variable expression  
*ValExpr*      A value expression

#### **► [Related topics](#)**

## **{AssignValue}**

### **Syntax**

{AssignValue *VarExpr*, *ValExpr*}

### **Description**

The {AssignValue} macro command is equivalent to the assignment statement *variable=value* in a programming language.

### **Tip**

- When you convert Quattro Pro macros to PerfectScript macros, the following Quattro Pro macro commands must be replaced by the corresponding PerfectScript commands:

<b>Quattro Pro</b>	<b>PerfectScript</b>
{ABS}	AbsoluteReference()
{ASSIGN}	AssignValue()

### **Example**

{AssignValue calc, CreateObject("DispCalc.Application")} creates an object of the DispCalc application and assigns it to a named variable calc.

{AssignValue calc.accum, 0} clears the accumulated value of DispCalc.

{AssignValue calc.accum, @SUM(A1..A10)} assigns the sum of A1..A10 to the accumulated value of DispCalc.

For more details on using {AssignValue} and other OLE automation macro commands, see [Using OLE Automation Features](#).

### **Parameters**

*VarExpr*      A variable expression  
*ValExpr*      A value expression

#### **► [Related topics](#)**



## **{Audit.Remove\_All\_Arrows}**

### **Description**

Removes all precedent and dependent arrows.

▶ **Related topics**

## **{Audit.Trace\_Dependents}**

### **Description**

Traces dependents of current formula.

▶ **Related topics**

## **{Audit.Trace\_Precedents}**

### **Description**

Traces precedents to current formula.

### **Tip**

- Equivalent to Tools ▶ Auditing
- ▶ Trace Precedents.
- ▶ **Related topics**



## **{BACKSPACE} and {BS}**

### **Syntax**

{BACKSPACE <*Number*>}

### **Description**

{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.

### **Parameters**

<i>Number</i>	Any positive integer or the address of a cell containing a positive integer (optional)
---------------	--

### **▶ Related topics**

## **{BACKTAB}**

### **Syntax**

{BACKTAB <*Number*>}

### **Description**

{BACKTAB} is equivalent to the Shift+Tab key. It is the same as [{BIGLEFT}](#), which moves one cell to the left.

When the Compatible Keys option in General Settings is checked, {BACKTAB} selects the leftmost cell of the screen that is 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.

### **Parameters**

*Number*  
*r*

Any positive integer or the address of a cell containing a positive integer (optional)

### **▶ Related topics**

## **{BEEP}**

### **Syntax**

`{BEEP <Number>}`

### **Description**

`{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 your attention. You can use it in interactive macros to introduce a prompt for information or to indicate a macro has finished.

### **Example**

The following macro checks a cell area named `error_check` for an error condition (indicated by `error_check` containing zero). If there is 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 cell area called `message_area`, where an error message is stored.

```
{IF error_check = 0}{BRANCH _continue}
{BEEP 1}{BEEP 5}{EditGoto message_area}
```

### **Parameters**

<i>Number</i>	1 to 10 (optional)
---------------	--------------------

### **► Related topics**

## **{BIGLEFT}**

### **Syntax**

{BIGLEFT <*Number*>}

### **Description**

{BIGLEFT} is equivalent to the Shift+Tab key. It is the same as [{BACKTAB}](#), which moves one cell to the left.

When the Compatible Keys option in General Settings is checked, {BIGLEFT} selects the leftmost cell of the screen that is to the left of the current cell.

The optional argument *Number* specifies how many times to repeat the operation; for example, {BIGLEFT 2} is equivalent to pressing Shift+Tab twice.

### **Parameters**

*Number*  
*r*

Any positive integer, or the address of a cell containing a positive integer (optional)

### **▶ Related topics**



# {BIGRIGHT}

## Syntax

{BIGRIGHT <*Number*>}

## Description

{BIGRIGHT}, like [{TAB}](#), is equivalent to the Tab key, which moves one cell to the right.

When the Compatible Keys option in General Settings is checked, {BIGRIGHT} selects the rightmost cell of the screen that is to the right of the current cell.

The optional argument *Number* specifies how many times to repeat the operation; for example, {BIGRIGHT 2} is equivalent to pressing Tab twice.

## Parameters

*Number*  
*r*

Any positive integer, or the address of a cell containing a positive integer (optional)

## ▶ [Related topics](#)

## **{BLANK}**

### **Syntax**

{BLANK *Location*}

### **Description**

{BLANK} erases the contents of the cells referred to as *Location*. You can also use the command equivalents [{ClearContents}](#) and [{EditClear}](#) to erase the contents of the currently selected cells.

### **Example**

This macro erases the cells named part\_list:

```
\F {BLANK part_list}
```

### **Parameters**

<i>Location</i>	Cell(s) you want erased
<i>n</i>	

### **► [Related topics](#)**

## {BlockCopy}

### Syntax

{BlockCopy *SourceBlock*, *DestBlock*, <*ModelCopy?*(0|1)>, <*Formula?*(0|1)>, <*Values?*(0|1)>, <*Properties?*(0|1)>, <*Objects?*(0|1)>, <*Row/Col\_Sizes?*(0|1)>, <*Labels?*(0|1)>, <*Numbers?*(0|1)>}

### PerfectScript Syntax

BlockCopy (SourceBlock:String; DestBlock:String; [ModelCopy?:Enumeration {Yes!; No!}]; [Formulas?:Enumeration {Yes!; No!}]; [Values?:Enumeration {Yes!; No!}]; [Properties?:Enumeration {Yes!; No!}]; [Objects?:Enumeration {Yes!; No!}]; [RowCol\_Sizes?:Enumeration {Yes!; No!}]; [Labels?:Enumeration {Yes!; No!}]; [Numbers?:Enumeration {Yes!; No!}])

### Description

{BlockCopy} copies the source cells to the specified destination. If *ModelCopy?* is 1, absolute references to cells within the copied cells adjust to reflect the new location. *Formula?*, *Values?*, *Properties?*, *Object?*, *Row/Col\_Sizes?*, *Labels?*, and *Numbers?* apply only if *ModelCopy?* is 1.

You can use {BlockCopy?} or {BlockCopy!} to display the Copy Cells dialog box. {BlockCopy?} lets you manipulate the dialog box, whereas {BlockCopy!} relies on the macro to manipulate it.

### Parameters

<i>SourceBlock</i>	Cells to copy
<i>DestBlock</i>	Location to copy cells
<i>ModelCopy?</i>	Whether to use Model Copy option; 0 = no, 1 = yes; the default is 0
<i>Formula?</i>	Whether to copy formula cells; 0 = no, 1 = yes; the default is 1
<i>Values?</i>	Whether to copy value cells; 0 = no, 1 = yes; the default is 1
<i>Properties?</i>	Whether to copy properties; 0 = no, 1 = yes; the default is 1
<i>Object?</i>	Whether to copy objects; 0 = no, 1 = yes; the default is 1
<i>Row/Col_Sizes?</i>	Whether to copy row and column sizes; 0 = no, 1 = yes; the default is 1
<i>Labels?</i>	Whether to copy label cells; 0 = no, 1 = yes; the default is 1
<i>Numbers?</i>	Whether to copy number cells; 0 = no, 1 = yes; the default is 1 (reserved for Cell Comments)

## {BlockDelete}

### Syntax

{BlockDelete.*Option*}

### PerfectScript Syntax

BlockDelete\_Columns (Block:String; Mode:Enumeration {Partial!; Entire!})

BlockDelete\_Pages (Block:String; Mode:Enumeration {Partial!; Entire!})

BlockDelete\_Rows (Block:String; Mode:Enumeration {Partial!; Entire!})

### Description

{BlockDelete.*Option*} deletes entire or partial columns, rows, and sheets. *Block* is the 2-D or 3-D selection where material is deleted.

You can use {BlockDelete?} or {BlockDelete!} to display the Delete dialog box. {BlockDelete?} lets you manipulate the dialog box, whereas {BlockDelete!} relies on the macro to manipulate it.

### Options

{BlockDelete.Columns <i>Block</i> , Entire Partial}	Deletes entire or partial column
{BlockDelete.Pages <i>Block</i> , Entire Partial}	Deletes entire or partial page

{BlockDelete.Rows *Block*, Entire|  
Partial}

Deletes entire or partial row

## {BlockFill}

### Syntax

{BlockFill.*Option*}

### PerfectScript Syntax

BlockFill\_Block (Block:String)

BlockFill\_Go ()

BlockFill\_Order (Order:Enumeration {Column!; Row!})

BlockFill\_Series (Series:Enumeration {Linear!; Growth!; Power!; Year!; Month!; Week!; Weekday!; Day!; Hour!; Minute!; Second!})

BlockFill\_Start (Value:Numeric)

BlockFill\_Step (Value:Numeric)

BlockFill\_Stop (Value:Numeric)

### Description

{BlockFill.*Option*} fills *Block* with sequential data. You can use numbers, dates, times, or even formulas for *Value*.

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 cells contains August.

You can enter the date and time directly as a serial number or use one of the date and time @functions.

You can use {BlockFill?} or {BlockFill!} to display the Fill Series dialog box. {BlockFill?} lets you manipulate the dialog box, whereas {BlockFill!} relies on the macro to manipulate it.

### Example

The following macro uses @DATEVALUE to enter 6/20/92 as the start value. The 3-D selection 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}
```

### Options

{BlockFill.Block <i>Block</i> }	Specifies the cells to fill with values.
{BlockFill.Go}	Fill the specified cells.
{BlockFill.Order Column Row}	Specifies whether to fill down columns or across rows.
{BlockFill.Series Linear   Growth   Power   Year   Month   Week   Weekday   Day   Hour   Minute   Second}	Specifies the type of fill operation to perform.
{BlockFill.Start <i>Value</i> }	Sets the first value in the series.
{BlockFill.Step <i>Value</i> }	Sets the constant value to add to the Start value or the last value.
{BlockFill.Stop <i>Value</i> }	Sets the limit for the fill values.

## {BlockInsert}

### Syntax

{BlockInsert.*Option*}

### PerfectScript Syntax

BlockInsert\_Columns (Block:String; Mode:Enumeration {Partial!; Entire!})

BlockInsert\_File (FileName:String; BeforeBlock:String)

BlockInsert\_Pages (Block:String; Mode:Enumeration {Partial!; Entire!})

BlockInsert\_Rows (Block:String; Mode:Enumeration {Partial!; Entire!})

### Description

{BlockInsert} inserts entire or partial columns, rows, and sheets, or complete files. *Block* is the 2-D or 3-D selection where material is inserted. In {BlockInsert.File}, *Filename* is inserted into the active notebook before *BeforeBlock*.

You can use {BlockInsert?} or {BlockInsert!} to display the Insert Cells dialog box. {BlockInsert?} lets you manipulate the dialog box, whereas {BlockInsert!} relies on the macro to manipulate it.

### Options

{BlockInsert.Columns <i>Block</i> , Entire Partial}	Inserts complete or partial columns.
{BlockInsert.File <i>FileName</i> , <i>BeforeBlock</i> }	Inserts a complete file.
{BlockInsert.Pages <i>Block</i> , Entire Partial}	Inserts complete or partial pages.
{BlockInsert.Rows <i>Block</i> , Entire Partial}	Inserts complete or partial rows.

## {BlockMove}

### Syntax

{BlockMove *SrcBlock*; *DstBlock*}

### PerfectScript Syntax

BlockMove (SrcBlock: String; DstBlock: String)

### Description

Lets you move a block.

### Parameters

<i>SrcBlock</i>	The block you want to move
<i>DstBlock</i>	New location for SrcBlock

### ▶ Related topics

## **{BlockMovePages}**

### **Syntax**

{BlockMovePages *SrcPages*, *BeforePage*}

### **Description**

{BlockMovePages} reorders sheets within a notebook. Moved sheets appear before *BeforePage*.

You can use {BlockMovePages?} or {BlockMovePages!} to display the Move Sheets dialog box.

{BlockMovePages?} lets you manipulate the dialog box, whereas {BlockMovePages!} relies on the macro to manipulate it.

### **Example**

The following macro will move the page named July to the position before the page named August.

```
{BlockMovePages July; August}
```

### **Parameters**

<i>SrcPages</i>	Range of sheets to move
<i>BeforePage</i>	New location for <i>SrcPages</i>

## {BlockName}

### Syntax

{BlockName.Option}

### PerfectScript Syntax

```
BlockName_AutoGenerate (Block:String; LabelsTop?:Enumeration {Yes!; No!};  
LabelsLeft?:Enumeration {Yes!; No!}; LabelsBottom?:Enumeration {Yes!; No!};  
LabelsRight?:Enumeration {Yes!; No!}; Intersection?:Enumeration {Yes!; No!})  
BlockName_Create (BlockName:String; Block:String)  
BlockName_Delete (BlockName:String)  
BlockName_Labels (Block:String; Where:Enumeration {Right!; Down!; Left!; Up!})  
BlockName_MakeTable (Block:String)  
BlockName_Reset ()
```

### Description

{BlockName} creates, deletes, and displays names for contiguous and noncontiguous selections.

*BlockName* is the cell name to create or delete. In {BlockName.Create}, *Block* refers to the cells to name; in {BlockName.MakeTable}, *Block* indicates where to create the name table. {BlockName.Reset} clears all cell names in the notebook.

You can use {BlockName?} or {BlockName!} to display the Cell Names dialog box. {BlockName?} lets you manipulate the dialog box, whereas {BlockName!} relies on the macro to manipulate it.

### Options

{BlockName.AutoGenerate <i>Block</i> , <i>LabelsTop?</i> (0 1), <i>LabelsLeft?</i> (0 1), <i>LabelsBottom?</i> (0 1), <i>LabelsRight?</i> (0 1), <i>Intersection?</i> (0 1)}	Creates cell names from adjacent labels.
{BlockName.Create <i>BlockName</i> , <i>Block</i> }	Adds a name for the specified cell to the cell name list.
{BlockName.Delete <i>BlockName</i> }	Deletes a selected cell name.
{BlockName.Labels <i>Block</i> ,Left Right Up Down}	Assigns names to single cells using adjacent labels.
{BlockName.MakeTable <i>Block</i> }	Creates a table in the notebook listing all named cells by name and location.
{BlockName.Reset}	Deletes all existing cell names from the notebook.

## {BlockReformat}

### Syntax

{BlockReformat *Block*}

### Description

{BlockReformat} adjusts word wrapping in a series of label entries (contained in *Block*) as though they were in a paragraph.

### Parameters

*Block*            The cells to reformat

## {BlockTranspose}

### Syntax

{BlockTranspose *SourceBlock*,*DestBlock*}

### Description

{BlockTranspose} copies *SourceBlock* to another location and reverses its rows and columns. Existing data in

*DestBlock* is overwritten.

You can use `{BlockTranspose?}` or `{BlockTranspose!}` to display the Transpose Cells dialog box. `{BlockTranspose?}` lets you manipulate the dialog box, whereas `{BlockTranspose!}` relies on the macro to manipulate it.

### Parameters

<i>SourceBlock</i>	Cells to transpose
<i>k</i>	
<i>DestBlock</i>	Cells to hold transposed copy

## {BlockValues}

### Syntax

`{BlockValues SourceBlock, DestBlock}`

### Description

`{BlockValues}` copies cells to another location and converts their formulas to values. Existing data in *DestBlock* is overwritten.

You can use `{BlockValues?}` or `{BlockValues!}` to display the Convert to Values dialog box. `{BlockValues?}` lets you manipulate the dialog box, whereas `{BlockValues!}` relies on the macro to manipulate it.

### Parameters

<i>SourceBlock</i>	Cells to copy as values
<i>k</i>	
<i>DestBlock</i>	Cells to hold converted copy

## {BRANCH}

### Syntax

`{BRANCH Location}`

### Description

`{BRANCH}` runs the macro stored in *Location*. If *Location* references cells, 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 does not hold your place in the original macro, waiting for control to return. Use `{BRANCH}` when you do not intend to return to the original macro. To run another macro and then return to the calling macro, use `{Subroutine}`.

### Example

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}
```

### Parameters

<i>Location</i>	Location or name of another macro
-----------------	-----------------------------------

### ► Related topics



## **{BREAK}**

### **Description**

{BREAK} clears any displayed dialog boxes or prompts and returns Quattro Pro to Ready mode. It does not stop macro execution; use [{QUIT}](#) for that operation.

### **► Related topics**

## **{BREAKOFF}**

### **Description**

{BREAKOFF} disables Ctrl+Break, which can be used to end a macro before it is done. After {BREAKOFF}, you will not be able to exit a macro until the end of the macro or until [{BREAKON}](#) is used.

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.

### **Example**

This macro disables Ctrl+Break while you input a name:

```
{PUTCELL "Enter your name here:"}
```

```
{BREAKOFF}
```

```
{?}~
```

```
{BREAKON}
```

```
{PUTCELL "Try again: "}
```

```
{?}~
```

### **▶ Related topics**

## **{BREAKON}**

### **Description**

{BREAKON} enables Ctrl+Break after a previous [{BREAKOFF}](#) command has disabled it.

Use {BREAKON} 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}.

▶ **Related topics**

## **{BudgetExpert}**

### **Description**

{BudgetExpert} displays the first Budget Expert dialog box. The macro has no arguments.



## **{CALC}**

### **Description**

{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.

### **▶ Related topics**

## **{CAPOFF} and {CAPON}**

### **Description**

{CAPOFF} and {CAPON} are equivalent to Caps Lock off and Caps Lock on, respectively.

▶ **Related topics**

## **{ChartExpert}**

### **Description**

{ChartExpert} displays the first Chart Expert dialog box. The macro has no arguments.



## **{CHOOSE}**

### **Description**

{CHOOSE} displays a pick list of open windows. Your choice becomes the active window.

▶ **Related topics**

## **{CLEAR}**

### **Description**

{CLEAR} is the equivalent of Ctrl+Backspace, which erases any previous entry in a prompt line or on the input line in Edit mode. This command is useful when loading or retrieving files.

### **▶ Related topics**

## **{ClearComments}**

### **Syntax**

{ClearComments <PageOnly? (0|1)>}

### **PerfectScript Syntax**

ClearComments ([PageOnly?:Numeric])

### **Description**

{ClearComments} deletes the comment in the active cell. PageOnly? flat refers to Group Mode. If Group mode is off, enter 0; if Group mode is on, and the active sheet belongs to a group, enter 1 to operate on only the active sheet or 0 to act on all sheets in the group. Equivalent to Rt-Clicking on the current cell, and choosing Delete Comment.

## **{ClearContents}**

### **Syntax**

{ClearContents <PageOnly?(0|1)>}

### **PerfectScript Syntax**

ClearContents ([PageOnly?:Enumeration {Yes!; No!}])

### **Description**

{ClearContents} erases the contents of the selected cells but leaves cell property settings intact.

### **Parameters**

<i>PageOnly</i>	If Group mode is off, enter 0; if Group mode is on, and the active sheet belongs to a group, enter 1 to operate on only the active sheet or 0 to act on all sheets in the group
?	

## **{ClearFormats}**

### **Syntax**

{ClearFormats <PageOnly?(0|1)>}

### **PerfectScript Syntax**

ClearFormats ([PageOnly?:Enumeration {Yes!; No!}])

### **Description**

{ClearFormats} resets the properties of cells but retains the values.

### **Parameters**

<i>PageOnly</i>	If Group mode is off, enter 0; if Group mode is on, and the active sheet belongs to a group, enter 1 to operate on only the active sheet or 0 to act on all sheets in the group
?	

### **▶ Related topics**

## **{CLOSE}**

### **Description**

{CLOSE} ends access to a file previously opened using {OPEN}. This lets you open another file (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.

### **Example**

The following macro opens a new file in drive A called AFILE, writes the text line Hello, world! to the file, and closes the file.

```
\F{OPEN "A:\AFILE",W}  
{WRITELN "Hello, world!"}  
{CLOSE}
```

# {COLUMNWIDTH}

## Syntax

{COLUMNWIDTH *Block*, *FirstPane?*, *Set/Reset/Auto*, *Size*}

## PerfectScript Syntax

ColumnWidth (Block:String; FirstPane?:Enumeration {Yes!; No!}; Mode:Enumeration {Set!; Reset!; Auto!}; Size:Numeric)

## Description

{COLUMNWIDTH} provides three ways to change the width of a column or columns (it is equivalent to the cell property Column Width). The columns to change are specified by *Block*. *FirstPane?* is used when the active window is split into 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 this 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 sheet Object Inspector) use this syntax: {COLUMNWIDTH *Block*, *FirstPane?*, 1}.

To automatically size a column based on what is entered in it, use this 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).

## Example

{COLUMNWIDTH A:A..B,1,0,1440} sets the width of columns A and B (on sheet A) to one inch (1,440 twips).

{COLUMNWIDTH A:A..B,0,0,2160} sets the width of columns A and B (on sheet 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 sheet A) to the default width.

{COLUMNWIDTH A:C,1,2,3} automatically sizes column C (on sheet A) and adds three characters to the calculated width.

## Parameters

<i>Block</i>	Cells containing columns to resize
<i>FirstPane?</i>	1 to resize columns in left or top window pane; 0 to resize columns in right or bottom window pane
<i>Set/Reset/Auto</i>	0 to set the column width; 1 to reset the column width; 2 to automatically size the column(s)
<i>Size</i>	New width (in twips) if Set/... = 0; not needed if Set/... = 1; resetting size; extra characters (optional) if Set/... = 2

## ► Related topics

## **{Comment.Edit}**

### **Syntax**

{Comment.Edit <CommentText>}

### **PerfectScript Syntax**

Comment\_Edit (Value?:String)

### **Description**

{Comment.Edit} creates/updates a comment in the active cell, and leaves comment "bubble" in edit mode for you to insert the comment text. If a comment already exists, it brings up the comment "bubble" in edit mode for you to edit the existing comment.

## **{Comment.EditURL}**

### **Syntax**

{Comment.EditURL <URLText>}

### **PerfectScript Syntax**

Comment\_EditURL (Value?:String)

### **Description**

Brings up the Insert Hyperlink dialog, allowing you to insert, modify, or delete a hyperlink.



## **{ComposeFormula}**

### **Description**

{Compose Formula} is the command equivalent of clicking the Formula Composer button on the Notebook toolbar. The macro has no arguments. {ComposeFormula} displays the Formula Composer dialog box.

# {Consolidate}

## Syntax

{Consolidate.*Option*}

## PerfectScript Syntax

Consolidate\_Add\_Source\_Block ([Block:String])

Consolidate\_Destination ([Block:String])

Consolidate\_Function (Function:Enumeration {SUM!; AVG!; COUNT!; MIN!; MAX!; STD!; STDS!; VAR!; VARS!})

Consolidate\_Go ()

Consolidate\_Options (OutputWithFormulas?:Enumeration {Yes!; No!}; LabelsInTopRow?:Enumeration {Yes!; No!}; LabelsInLeftCol?:Enumeration {Yes!; No!})

Consolidate\_Remove (Name:String)

Consolidate\_Remove\_Source\_Block ([Block:String])

Consolidate\_Reset ()

Consolidate\_Save (Name:String)

Consolidate\_Use (Name:String)

## Description

{Consolidate} combines data from multiple selections into one using your choice of operators. *Block* defaults to the current selection if the argument is not supplied.

You can use {Consolidate?} or {Consolidate!} to display the Consolidation dialog box. {Consolidate?} lets you manipulate the dialog box, whereas {Consolidate!} relies on the macro to manipulate it.

## Example

The following macro adds the values in the source cells B2..B4, C2..C3, and D2..D4, and returns values in the destination cells F2..F4.

```
{Consolidate.Add_Source_Block A:B2..B4}  
{Consolidate.Add_Source_Block A:C2..C3}  
{Consolidate.Add_Source_Block A:D2..D4}  
{Consolidate.Function SUM}  
{Consolidate.Destination A:F2..F4}  
{Consolidate.Options 1,0,0}  
{Consolidate.Go}  
{Consolidate.Save CONSOL1}
```

## Options

{Consolidate.Add_Source_Block <Block>}	Adds an entry to the Source Cells list.
{Consolidate.Destination <Block>}	Sets the cells to contain the consolidation results.
{Consolidate.Function <i>SummaryFunction</i> }	Specifies the operations to perform on the source cells.
{Consolidate.Go}	Performs the consolidation of the source cells.
{Consolidate.Options <i>OutputWithFormulas?</i> (0 1), <i>LabelsInTopRow?</i> (0 1), <i>LabelsInLeftCol?</i> (0 1)}	Selects options for consolidation.
{Consolidate.Remove <i>Name</i> }	Deletes the selected setup in the Consolidations list.
{Consolidate.Remove_Source_Block <Block>}	Removes an entry from the Source Cells list.
{Consolidate.Reset}	Clears Source Cells and Destination Cells, and resets Options to default values in the Consolidation dialog box..
{Consolidate.Save <i>Name</i> }	Saves the current consolidation setup.
{Consolidate.Use <i>Name</i> }	Lists saved consolidation setups.

# {ConsolidateExpert}

**Description**

{ConsolidateExpert} displays the first Consolidate Expert dialog box. The macro has no arguments.

## {CONTENTS}

### Syntax

{CONTENTS *Dest*, *Source*, <*Width#*>, <*Format#*>}

### PerfectScript Syntax

Contents (DestCell:String; SourceCell:String; [Width:Numeric]; [Format:Numeric])

### Description

{CONTENTS} copies the contents of *Source* into *Dest*, but unlike [{LET}](#) or other copy commands, if *Source* contains a value entry, it translates the copied value into a label and stores it in *Dest*. 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 will not 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 is not 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 do not 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. *Format#* affects the *Dest* entry only, not the *Source* value. See [Numeric Format Codes](#) for a list of special codes used to indicate numeric formats with *Format#*.

### Example

The following examples assume cell C18 contains the value 48,988 in comma format with a column width of 12.

{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 does not 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).

### Parameters

<i>Dest</i>	Cell you want data written to
<i>Source</i>	Cell you want data copied from
<i>Width#</i>	Optional column width (1 to 1023)
<i>Format#</i>	Optional format code
<i>#</i>	

## {Controls}

### Syntax

{Controls.*Option*}

### PerfectScript Syntax

Controls\_Order ()

Controls\_OrderFrom ()

Controls\_OrderTab ()

Controls\_OrderTabFrom ()

### Description

{Controls} affects selected objects in the dialog window.

### Options

{Controls.Order}	Changes the setting order of controls
{Controls.OrderFrom}	Places related controls together in the setting order
{Controls.OrderTab}	Sets the tab order for controls

```
{Controls.OrderTabFrom  
m}
```

Pulls specific controls out of the tab order and groups them together

▶ **Related topics**

**{CR} or ~**

**Description**

{CR} or ~ (tilde) are equivalent to the Enter key.

▶ **Related topics**

## **{CreateChart}**

### **Syntax**

{CreateChart *Name*}

### **PerfectScript Syntax**

CreateChart (Name: String)

### **Description**

Lets you create a chart.

### **Parameter**

*Name*

The name of the chart

### **▶ Related topics**

## {CREATEOBJECT}

### Syntax

{CREATEOBJECT *ObjectType*, *x1*, *y1*, *x2*, *y2*<, *x3*, *y3*, ...>}

### PerfectScript Syntax

CreateObject (ObjectName:String; x1:Numeric; y1:Numeric; x2:Numeric; y2:Numeric; {[x:Numeric]; [y:Numeric]})

### Description

With {CREATEOBJECT} you can add objects to the active window normally added using the Toolbar. {CREATEOBJECT} is context-sensitive, letting you create lines in a chart window or check boxes in a dialog window. Quattro Pro interprets the coordinates specified after *ObjectType* differently based on the object type. The following table lists the possible chart object settings for *ObjectType*, and how each chart object uses the (x,y) coordinates.

### Chart Objects {CREATEOBJECT} Can Generate

Object	# of (x,y)'s	Coordinates
Line	2	1st: Start point, 2nd: End point
Arrow		(same as for Line)
Block	2	1st: Upper left corner, 2nd: Width and height of the objects (in relative coordinates)
Rect (Rectangle)	2	(same as for Block)
Ellipse	2	1st: Upper left corner of a rectangle bounding the ellipse; 2nd: Width and height of the bounding rectangle
Rounded_Rect		(same as for Block)
Text		(same as for Block)
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, ... <i>n</i> th: End point
Polygon		(same as for Polyline)
Freehand_Polyline		(same as for Polyline)
Freehand_Polygon		(same as for Polyline)

### Block Objects

The Block object has additional arguments for {CREATEOBJECT}:

{CREATEOBJECT *ObjectType*, *x1*, *y1*, *x2*, *y2*, "Block", "RowBorders?(Yes|No), ColBorders?(Yes|No), HorzGridLines?(Yes|No), VertGridLines?(Yes|No), AspectRatio?(Yes|No)"}

*Block* sets the notebook cells to use. The remaining arguments specify whether to show borders and grid lines and whether to maintain the cells' aspect ratio.

### Dialog Controls {CREATEOBJECT} Can Generate

You can create these dialog controls listed in the order they appear on the Dialog Toolbar: 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.

*ObjectType* is enclosed in quotes. The x and y coordinates for each point follow, separated by commas.

### Example

{CREATEOBJECT "Rect",86,11,94,74} creates a rectangle with upper-left corner = (86,11), width = 94, and height = 74 (pixels).

{CREATEOBJECT "Block", 363, 260, 1278, 1139, "A:B2..D9", "No,No,Yes,Yes,Yes"} creates notebook cells in a chart window with upper-left corner = (363, 260), width = 1278, and height = 1139; the other arguments specify the notebook cells, turn off row and column borders, show grid lines, and maintain the cells' aspect ratio.

{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).



## Parameters

<i>ObjectNam</i>	Type of object to create
<i>x1, y1</i>	XY coordinates for the starting point of the object; the upper left corner for rectangles and objects bounded by rectangles
<i>x2, y2</i>	XY coordinates for the end point or next point of the object; the width and height for rectangles and objects bounded by rectangles
<i>x3, y3</i>	XY coordinates for the next or last point of a polyline or polygon object

### ► Related topics

## {CrossTab}

### Syntax

```
{CrossTab "Input Cells";"Output cells";"<3D Page Name>";"Row 1;<Row 2>;<Row 3>";"Column 1;<Column 2>;<Column 3>";"Data 1: Data Option,<Data 2: Data Option>";"<Row 1: Option>,<Row 2: Option>,<Row 3: Option>,<Column 1: Option>,<Column 2: Option>,<Column 3: Option>"}
```

### PerfectScript Syntax

```
CrossTab (SrcBlock:String; DstBlock:String; PageName:String; RowData:String; ColData:String; {[DataTotal:String]})
```

### Description

{CrossTab} creates a summary of your data in a format that is simple and easy to read. This is especially useful when you are working with large pieces of data, such as imported databases.

All items surrounded by <> are optional. All quotes in this macro command must be included in order for the macro to function.

All Column, Row and Data items are to be replaced with the field number containing the data to be used. Fields go from 0 to however many columns are passed into Cross Tabs. Columns are numbered from left to right in the source range, 0 being the first column of the selection.

### Example

```
{CrossTab "A:A1..H145";"B:A1";"";"0,1";"2,3,4";"6: SUM";"4: AVERAGE"}
```

Notice that if the 3D Sheet Name is not included, the macro must have the empty quotes or it will not function properly.

### Parameters

<i>Data Option</i>	SUM, AVERAGE, COUNT, % of COLUMN, % of ROW, % of GRAND, or STRING
<i>Row and Column Options</i>	SUM, AVERAGE, COUNT, % of COLUMN, % of ROW, % of GRAND, INCREASE, % INCREASE, or STRING

## {CrossTabReport.AddField}

### Syntax

```
{CrossTabReport.AddField Index; Type}
```

### PerfectScript Syntax

```
CrossTabReport_AddField (Index?: Numeric; Type?: Numeric)
```

### Description

Lets you add a field to the active report.

### Parameters

<i>Index</i>	The index of the field 1 Row 2 Column 3 Page 4 Data
<i>Type</i>	

### Example

A [sample](#) Cross Tab Report has the following macro commands run against it.

```
{CrossTabReport.AddField 3;3}
```

```
{CrossTabReport.Edit}
```

The [result](#) is that the Winery field (index position 3 in the underlying data source) has been added to the page area of the Report.

## **{CrossTabReport.CenterLabels}**

### **Syntax**

`{CrossTabReport.CenterLabels Enable}`

### **PerfectScript Syntax**

`CrossTabReport_CenterLabels (Enable?: Boolean)`

### **Description**

Lets you specify whether or not to center the labels on a report.

### **Parameter**

<i>Enable</i>	0 Do not center the labels
	1 Center the labels.

### **Example**

A sample Cross Tab Report has the following macro commands run against it.

```
{CrossTabReport.CenterLabels 1}
```

```
{CrossTabReport.Options}
```

The result is that the Year labels (1991 and 1992) have been centered against the rows of data.

## **{CrossTabReport.ColumnSummary}**

### **Syntax**

{CrossTabReport.ColumnSummary *Enable*}

### **PerfectScript Syntax**

CrossTabReport\_ColumnSummary (Enable?: Boolean)

### **Description**

Lets you specify whether or not to display a column summary.

### **Parameter**

<i>Enable</i>	0 Do not display a column summary.
	1 Display a column summary.

### **Example**

A sample Cross Tab Report has the following macro commands run against it.

```
{CrossTabReport.ColumnSummary 1}
```

```
{CrossTabReport.Options}
```

The result is that each of the columns of sales data (Q1-Q4) have been added together and a grand total displayed at the bottom of each.

## **{CrossTabReport.CopyStatic}**

### **Syntax**

{CrossTabReport.CopyStatic}

### **PerfectScript Syntax**

CrossTabReport\_CopyStatic ()

### **Description**

A command macro which creates a static copy of the current Cross Tab Report. The copy does not hold any properties of the report and is not affected by changes in the underlying source data.

## **{CrossTabReport.Create}**

### **Syntax**

{CrossTabReport.Create}

### **PerfectScript Syntax**

CrossTabReport\_Create ()

### **Description**

A command macro which is used to generate a new Cross Tab Report. As shown below, this macro is typically used in conjunction with the {CrossTabReport.Source}, {CrossTabReport.Destination}, {CrossTabReport.Name}, and {CrossTabReport.AddField} macros.

### **Example**

A sample spreadsheet is used as the data source for a Cross Tab Report. To create the report, the following sequence of macro commands is run.

```
{CrossTabReport.Source A:A1..H145}  
{CrossTabReport.Destination B:A1}  
{CrossTabReport.Name CrossTabs Table 1}  
{CrossTabReport.AddField 1;1}  
{CrossTabReport.AddField 2;2}  
{CrossTabReport.AddField 8;4}  
{CrossTabReport.Create}
```

The result is that a new Cross Tab Report is created. It uses columns A through H in Sheet A as its data source, and cell A1 in Sheet B is used as the destination for the report. The Year, Quarter, and Sales fields are then added to the Cross Tab Report's row, column, and data areas respectively.

## **{CrossTabReport.DataAlignment}**

### **Syntax**

`{CrossTabReport.DataAlignment RowOrCol}`

### **PerfectScript Syntax**

`CrossTabReport_DataAlignment (RowOrCol?: Numeric)`

### **Description**

Lets you specify whether the data fields in a report are aligned by row or column. By default, data fields are aligned in a row.

### **Parameter**

<i>RowOrCol</i>	0 Row
	1 Column

### **Example**

A [sample](#) Cross Tab Report has its data fields (Sales and Cost Per Case) aligned by row. To change this, the following macro commands are run.

```
{CrossTabReport.DataAlignment 1}  
{CrossTabReport.Edit}
```

The [result](#) is a Cross Tab Report which now has its data fields aligned by column.

## **{CrossTabReport.DefineFieldProps}**

### **Syntax**

`{CrossTabReport.DefineFieldProps Area; Index}`

### **PerfectScript Syntax**

`CrossTabReport_DefineFieldProps (Area?: String ; Index: String)`

### **Description**

Lets you specify the fields on which specified options will operate. Typically, this macro will be followed by other macros which perform the desired operation on the specified field. For example, the `{CrossTabReport.FieldSummary}` and `{CrossTabReport.FieldOptions}` macros might be used, as shown below, to specify which operations to perform on the specified field.

### **Parameter**

<i>Area</i>	1 Row area 2 Column area 3 Page area 4 Data area
<i>Field Index</i>	The index of the given field based on its position in that area

### **Example**

A sample Cross Tab Report contains two data fields, Sales and Cases Sold, both of which already have the summary option Sum. To add the summary option Max to only the Sales field, and not the Cases Sold field, the following macro commands are run

```
{CrossTabReport.DefineFieldProps "4;1"}
```

```
{CrossTabReport.FieldSummary "1;4"}
```

```
{CrossTabReport.FieldOptions}
```

The result is that the first field in the data area (Sales) is defined as the field on which to apply the summary option Max. For more information on the options applied to the defined field, refer to the help for the macros `{CrossTabReport.FieldSummary}` and `{CrossTabReport.FieldOptions}`.



## **{CrossTabReport.Destination}**

### **Syntax**

{CrossTabReport.Destination *Block*}

### **PerfectScript Syntax**

CrossTabReport\_Destination (Block?: String)

### **Description**

Lets you specify where the report is located.

### **Parameter**

<i>Block</i>	The destination cell
--------------	----------------------

### **Example**

When creating a Cross Tab Report, the following macro command is used to specify a destination cell for the report.

```
{CrossTabReport.Destination B:A1}
```

The result is a Cross Tab Report residing on sheet B, cell A1. For a more detailed example, refer to the help for the [{CrossTabReport.Create}](#) macro.

## **{CrossTabReport.DisplayInEmptyCell}**

### **Syntax**

{CrossTabReport.DisplayInEmptyCell *Enable*}

### **PerfectScript Syntax**

CrossTabReport\_DisplayInEmptyCell (Enable?: Boolean)

### **Description**

Lets you specify whether or not to display a specific value in the empty cells of a report.

### **Parameter**

<i>Enable</i>	0 Do not display a value in empty cells.
	1 Display a value in empty cells.

### **Example**

A sample report contains one or more cells which are empty or awaiting future data. To fill these cells with some value, say "TBA", the following macro commands are used.

```
{CrossTabReport.DisplayInEmptyCell 1}  
{CrossTabReport.EmptyCellString TBA}  
{CrossTabReport.Options}
```

The result is a Cross Tab Report with the value TBA displayed in any previously empty cells. Note that the {CrossTabReport.EmptyCellString} can be used to specify the text which will appear in place of the empty cell.

## **{CrossTabReport.Edit}**

### **Syntax**

{CrossTabReport.Edit}

### **PerfectScript Syntax**

CrossTabReport\_Edit ()

### **Description**

A command macro which is used to modify the report settings or configuration. Typically, this macro is used after a sequence of operations such as adding a field or changing the destination of a report.

### **Example**

For an example detailing the usage of the {CrossTabReport.Edit} macro, see the help for either the [{CrossTabReport.AddField}](#) macro or the [{CrossTabReport.DataAlignment}](#) macro.

## **{CrossTabReport.EmptyCellString}**

### **Syntax**

{CrossTabReport.EmptyCellString *Name*}

### **PerfectScript Syntax**

CrossTabReport\_EmptyCellString (Name?: String)

### **Description**

Lets you specify the string to be displayed in the empty cells of a Cross Tab Report.

### **Parameter**

<i>Name</i>	The string to be displayed in empty cells
-------------	---

### **Example**

A sample report contains one or more cells which are empty or awaiting future data. To fill these cells with some value, say "TBA" the following macro commands are used.

```
{CrossTabReport.DisplayInEmptyCell 1}
```

```
{CrossTabReport.EmptyCellString TBA}
```

The result is a Cross Tab Report with the value TBA displayed in any previously empty cells. Note that the {CrossTabReport.DisplayInEmptyCell}.is used to specify whether or not a value is displayed in empty cells.

## {CrossTabReport.Expand}

### Syntax

```
{CrossTabReport.Expand <Index> <;Index2>}
```

### PerfectScript Syntax

```
CrossTabReport ([Index?: Numeric] [;Index2?: Numeric])
```

### Description

Lets you expand the current report onto several different sheets by specifying the appropriate field indices. By default, this macro command will expand to the maximum number of levels. Note that in order to use this macro, you must have a least one field in the Pages position of the report.

### Parameters

<i>Index1</i>	The field on which you want to base the report expansion.
<i>Index2</i> [optiona l]	The number of levels to which you want to expand the report.

### Example

A sample report, located on sheet A of a notebook, contains two fields in the Pages area of the report. The field "Winery", located in index position 1, contains two field items, Beaulieu and Duckhorn. To expand the report based on the items in this field, the following macro commands are used

```
{CrossTabReport.Expand 1}
```

The result is that the Cross Tab Report is expanded onto the next two unprotected pages in the notebook; in this case sheet B and sheet C. Sheet B contains the field item Beaulieu and all the data associated with it, and sheet C contains the field item Duckhorn and all the data associated with it.

## **{CrossTabReport.FieldCmp}**

### **Syntax**

{CrossTabReport.FieldCmp *Value*}

### **PerfectScript Syntax**

CrossTabReport\_FieldCmp (Value?: Numeric)

### **Description**

Lets you specify a comparison option on any given field within a report. Typically, this macro will be used along with the [{CrossTabReport.FieldCmpBase}](#), [{CrossTabReport.FieldCmpltem}](#), and [{CrossTabReport.FieldCmpltemPreset}](#) macros.

### **Parameter**

<i>Value</i>	
0	None
1	DiffFrom
2	PercentOf
3	PercentDiffFrom
4	RunningTotal
5	PercentRow
6	PercentColumn
7	PercentTotal
8	Index

### **Example**

A [sample](#) Cross Tab Report has the following macro commands run against it.

```
{CrossTabReport.DefineFieldProps "4,1"}  
{CrossTabReport.FieldCmp 1}  
{CrossTabReport.FieldCmpBase 1}  
{CrossTabReport.FieldCmpItemPreset -1}  
{CrossTabReport.FieldOptions}
```

The [{CrossTabReport.DefineFieldProps}](#) macro is used to indicate that the specified comparison options are to be applied to the Sales field in the Data area of the page. The [result](#) is a report which takes the sales data in each row of the Year field (index position 1) and calculates the difference between it and the data from the previous year.

## **{CrossTabReport.FieldCmpBase}**

### **Syntax**

{CrossTabReport.FieldCmpBase *Value*}

### **PerfectScript Syntax**

CrossTabReport\_FieldCmpBase (Value?: Numeric)

### **Description**

Lets you specify the index of the base field.

### **Parameter**

<i>Value</i>	The index of the base field
--------------	-----------------------------

### **Example**

```
{CrossTabReport.Field CmpBase 1}
```

The field with index value 1 is taken to be the base field for comparison. For a more detailed example involving this macro, please see the help for the [{CrossTabReport.FieldCmp}](#) macro.

## **{CrossTabReport.FieldCmplItem}**

### **Syntax**

{CrossTabReport.FieldCmplItem *Value*}

### **PerfectScript Syntax**

CrossTabReport\_FieldCmplItemPreset (Value?: String)

### **Description**

Lets you specify the field item to be compared.

### **Parameter**

<i>Value</i>	The index of the field
--------------	------------------------

### **Example**

```
{CrossTabReport.FieldCmpItem 2}
```

The field with index value 2 is defined as the item to be compared.



## **{CrossTabReport.FieldCmplItemPreset}**

### **Syntax**

{CrossTabReport.FieldCmplItemPreset *Value*}

### **PerfectScript Syntax**

CrossTabReport\_FieldCmplItemPreset (Value?: Numeric)

### **Description**

Lets you specify the type of preset to be used during comparison.

### **Parameter**

<i>Value</i>	0 None
	-1 Previous
	1 Next

### **Example**

```
{CrossTabReport.FieldCmpItemPreset -1}
```

Previous is selected as the type of preset to be used during the comparison. For a more detailed example involving this macro, see the help for the [{CrossTabReport.FieldCmp}](#) macro.

## {CrossTabReport.FieldHide}

### Syntax

```
{CrossTabReport.FieldHide Value}
```

### PerfectScript Syntax

```
CrossTabReport_FieldHide (Value?: String)
```

### Description

Lets you hide one or more data items associated with the report. You can specify the field by using the [{CrossTabReport.DefineFieldProps}](#) command.

### Parameter

<i>Value</i>	Semicolon delimited items
[semicolon n delimited]	

### Example

A [sample](#) Cross Tab Report has the following macro commands run against it.

```
{CrossTabReport.DefineFieldProps "1;1"}  
{CrossTabReport.FieldHide "1991"}  
{CrossTabReport.FieldOptions}
```

The [result](#) is a report which hides the field item 1991 and its data.

#### ► Note

- You can leave the *Value* value empty to clear the existing values.

## **{CrossTabReport.FieldLabel}**

### **Syntax**

`{CrossTabReport.FieldLabel Value}`

### **PerfectScript Syntax**

`CrossTabReport_FieldLabel (Value?: String)`

### **Description**

Lets you specify or change the label on a given field. You can specify the field by using the [{CrossTabReport.DefineFieldProps}](#) command.

### **Parameter**

<i>Value1</i>	Text for the field label
---------------	--------------------------

### **Example**

A [sample](#) Cross Tab Report has the following macro commands run against it.

```
{CrossTabReport.DefineFieldProps "1,1"}
```

```
{CrossTabReport.FieldLabel Years}
```

```
{CrossReport.FieldOptions}
```

The [result](#) is that the label which previously displayed as "Year", has been modified to display as "Years".

## **{CrossTabReport.FieldOptions}**

### **Syntax**

{CrossTabReport.FieldOptions}

### **PerfectScript Syntax**

CrossTabReport\_FieldOptions ()

### **Description**

This is a command macro used to modify field options. Typically, macro operations which modify a field will be followed by this command macro.

### **Example**

For examples detailing the usage of this macro, refer to the help for either the [{CrossTabReport.FieldCmp}](#) [{CrossTabReport.FieldHide}](#) macros.

## {CrossTabReport.FieldSummary}

### Syntax

{CrossTabReport.FieldSummary *Value*}

### PerfectScript Syntax

CrossTabReport\_FieldSummary (Value?: String)

### Description

Lets you specify one or more summary option flags. *Value* consists of variables delimited by semicolons. You can specify the field by using the [{CrossTabReport.DefineFieldProps}](#) command.

### Parameter

<i>Value</i>	1 Sum
[semicol	2 Count
on	3 Average
delimite	4 Max
d]	5 Min
	6 StdDevp
	7 StdDevs
	8 Varp
	9 Var
	10 CountNonBlank
	11 SumNone (clears existing flags)

### Example

A [sample](#) Cross Tab Report has the following macro commands run against it.

```
{CrossTabReport.DefineFieldProps "4;1"}
```

```
{CrossTabReport.FieldSummary "1; 3; 4; 5"}
```

```
{CrossTabReport.FieldOptions}
```

The [result](#) is a report which now calculates and displays Sum, Average, Max, and Min values for the Sales field.

## **{CrossTabReport.FormatReport}**

### **Syntax**

{CrossTabReport.FormatReport *Enable*}

### **PerfectScript Syntax**

CrossTabReport\_FormatReport (Enable?: Boolean)

### **Description**

Lets you specify whether or not to apply a predefined format to the report.

### **Parameter**

<i>Enable</i>	0 Do not apply a predefined format to the report.
	1 Apply a predefined format to the report.

### **Example**

A [sample](#) Cross Tab Report, with a predefined format applied, has the following macro commands run against it.

```
{CrossTabReport.FormatReport 0}
```

```
{CrossTabReport.Options}
```

The [result](#) is a report which no longer has a predefined format applied. Note that in this example the dark cell borders have been lost as a result of the predefined format no longer being applied.

## **{CrossTabReport.Hide}**

### **Syntax**

{CrossTabReport.Hide}

### **PerfectScript Syntax**

CrossTabReport\_Hide ()

### **Description**

A command macro used to hide the details of the active or selected field in a report.

### **Example**

Within a [sample](#) Cross Tab Report, the active cursor selection is positioned within the Q1 field item and the following macro command is executed.

```
{CrossTabReport.Hide}
```

The [result](#) is a report which displays without any details for Q1. All other field items continue to display as they were originally.

### **► Note**

The [{CrossTabReport.Show}](#) macro can be used to return the report to its original state.

## **{CrossTabReport.LabelEdit}**

### **Syntax**

{CrossTabReport.LabelEdit *LabelEdit*}

### **PerfectScript Syntax**

CrossTabReport\_LabelEdit (LabelEdit?: String)

### **Description**

Lets you change the label of the selected field cell in the sheet. This macro allows you to edit a field label from the active report without going through the field options.

### **Parameter**

*LabelEdit*      The changed label of the selected field cell

### **Example**

Within a sample Cross Tab Report, the active cursor selection is positioned at the label to be changed and the following macro command is executed.

```
{CrossTabReport.LabelEdit Years}
```

The result is that the label which previously displayed as "Year", has been modified to display as "Years".



## **{CrossTabReport.MoveCell}**

### **Syntax**

`{CrossTabReport.MoveCell Row; Column}`

### **PerfectScript Syntax**

`CrossTabReport_MoveCell (Row?: Numeric; Column?: Numeric)`

### **Description**

Lets you move a selected cell within the active report to a specified destination cell.

### **Parameters**

<i>Row</i>	The row you to which you want to move the selected cell.
<i>Column</i>	The column to which you want to move the selected cell.

### **Example**

Within a [sample](#) Cross Tab Report, the active cursor selection is positioned at the Year label cell, and the following macro command is executed.

```
{CrossTabReport.MoveCell 1;3}
```

The [result](#) is a report in which the Year field is now displayed in the column area instead of the row area as it had been previously.

## {CrossTabReport.MoveField}

### Syntax

{CrossTabReport.MoveField *Source\_Index*; *Source\_Type*; *Destination\_Index*; *Destination\_Type*}

### PerfectScript Syntax

CrossTabReport\_MoveField (Source\_Index?: Numeric; Source\_Type?: Numeric; Destination\_Index?: Numeric; Destination\_Type?: Numeric)

### Description

Lets you move the selected field to a new position within an active report.

### Parameters

<i>Source_Area</i>	1 Row area 2 Column area 3 Page area 4 Data area
<i>Source_Index</i>	The numeric index of the source field
<i>Destination_Area</i>	1 Row area 2 Column area 3 Page area 4 Data area
<i>Destination_Index</i>	The numeric index of the destination field

### Example

Within a [sample](#) Cross Tab Report, the active cursor selection is positioned in the Year field, and the following macro command is executed.

```
{CrossTabReport.MoveField 1;1;2;2}  
{CrossTabReport.FieldOptions}
```

The [result](#) is a report in which the Year field is now displayed in the column area instead of the row area as it had been previously. The field has been moved from index position 1 of the Row area to index position 2 of the Column area.

## **{CrossTabReport.Name}**

### **Syntax**

{CrossTabReport.Name *Name*}

### **PerfectScript Syntax**

CrossTabReport\_Name (Name?: String)

### **Description**

Lets you specify or change the name of an active report.

### **Parameter**

<i>Name</i>	The name of the report
-------------	------------------------

### **Example**

A report is named CrossTabs Table 1. To change this, the following macro command is executed.

```
{CrossTabReport.Name "CrossTabs Table 2"}
```

The report is now named CrossTabs Table 2. The new report name can be viewed or verified using the Cross Tabs Options dialog box.

## **{CrossTabReport.Options}**

### **Syntax**

{CrossTabReport.Options}

### **PerfectScript Syntax**

CrossTabReport\_Options ()

### **Description**

A command macro used to modify the report options. Typically, this macro is used after a sequence of commands such as showing a column summary or displaying a value in empty cells.

### **Example**

For an example detailing the usage of the {CrossTabReport.Options} macro, see the help for either the [{CrossTabReport.ColumnSummary}](#) macro or the [{CrossTabReport.DisplayInEmptyCell}](#) macro

## **{CrossTabReport.PageFilter}**

### **Syntax**

{CrossTabReport.PageFilter *Index*; *Value*}

### **PerfectScript Syntax**

CrossTabReport\_PageFilter (Index?: Numeric; Value?: String)

### **Description**

Lets you apply a page filter to the specified field and value in the page area.

### **Parameters**

<i>Index</i>	The numeric index of the field to be filtered.
<i>Value</i>	The field value on which you filter the report.

### **Example**

A [sample](#) Cross Tab Report has a Winery field located in the Pages area. This field contains the items Beaulieu, Duckhorn, and [All]. To filter this report, the active cursor selection is positioned within the report, and the following macro command is executed.

```
{CrossTabReport.PageFilter 1; Duckhorn}
```

The [result](#) is a report which has been filtered on index position 1 of the Pages area. In this example, the report now shows only data relating to the Duckhorn winery.

## **{CrossTabReport.PreserveDataFormat}**

### **Syntax**

{CrossTabReport.PreserveDataFormat *Enable*}

### **PerfectScript Syntax**

CrossTabReport\_PreserveDataFormat (Enable?: Boolean)

### **Description**

Lets you specify whether or not the report should preserve the formatting options found in the data source.

### **Parameter**

<i>Enable</i>	0 Do not preserve the data format from source
	1 Preserve the data format from source.

### **Example**

A [sample](#) report has been generated without retaining the source data formatting. In the source, all data had appeared in bold. To apply the source data formatting to the report, the following macro command is executed.

```
{CrossTabReport.PreserveDataFormat 1}
```

```
{CrossTabReport.Edit}
```

The [result](#) is a report which now applies the formatting options found in the source. All data now appears in bold.

## **{CrossTabReport.Refresh}**

### **Syntax**

{CrossTabReport.Refresh}

### **PerfectScript Syntax**

CrossTabReport\_Refresh ()

### **Description**

A command macro that lets you refresh the active report to reflect changes in the source data.

### **Example**

A sample report is generated from a data source. Now suppose the underlying data source is changed, for example, to reflect an increase in sales of \$20,000 for Q4 of 1992. In order to have this change reflected in the report, the following macro command is executed while the active cursor selection is within the report.

```
{CrossTabReport.Refresh}
```

The result is a report updated to reflect changes in the source data. Note that the figure in Q4 of 1992 has changed.

## **{CrossTabReport.Remove}**

### **Syntax**

{CrossTabReport.Remove}

### **PerfectScript Syntax**

CrossTabReport\_Remove ()

### **Description**

A command macro which removes the active report.



## **{CrossTabReport.RowSummary}**

### **Syntax**

{CrossTabReport.RowSummary *Enable*}

### **PerfectScript Syntax**

CrossTabReport\_RowSummary (Enable?: Boolean)

### **Description**

Lets you specify whether or not to display row summaries in a report.

### **Parameter**

<i>Enable</i>	0 Do not display row summaries for the report.
	1 Display row summaries for the report

### **Example**

A sample report summarizes sales data. To add a row summary which calculates the total sales for each year, the following macro commands are executed.

```
{CrossTabReport.RowSummary 1}  
{CrossTabReport.Options}
```

The result is that each of the rows of sales data (1991-1992) have been added together and a grand total displayed at the end of each.

## **{CrossTabReport.Show}**

### **Syntax**

{CrossTabReport.Show}

### **PerfectScript Syntax**

CrossTabReport\_Show ()

### **Description**

A command macro used to show the details of the active or selected field in a report.

### **Example**

Within a [sample](#) Cross Tab Report, the active cursor selection is positioned within the Q1 field item, which has its details hidden, and the following macro command is executed.

```
{CrossTabReport.Hide}
```

The [result](#) is a report which displays the details for Q1. All other field items continue to display as they were originally.

### **► Note**

The [{CrossTabReport.Hide}](#) macro can be used to return the report to its original state.

## **{CrossTabReport.Source}**

### **Syntax**

{CrossTabReport.Source *Block*}

### **PerfectScript Syntax**

CrossTabReport\_Source (Block?: String)

### **Description**

Lets you specify the sheet and range of cells from which you want to generate the report.

### **Parameter**

*Block*            The range of cells.

### **Example**

When creating a Cross Tab Report, the following macro command is used to specify the source for the report.

```
{CrossTabReport.Source A:A1..H145}
```

The report is generated from cells A1 to H145 on sheet A. For a more detailed example involving this macro, refer to the help for the [{CrossTabReport.Create}](#) macro.

## **{CrossTabReport.UpdateDataOnOpen}**

### **Syntax**

{CrossTabReport.UpdateDataOnOpen *Enable*}

### **PerfectScript Syntax**

CrossTabReport\_UpdateDataOnOpen (Enable?: Boolean)

### **Description**

Lets you specify whether or not to update data when you open the report.

### **Parameter**

<i>Enable</i>	0 Do not update the data.
	1 Update the data.

### **Example**

To update a report upon opening it, the following macro command is executed.

```
{CrossTabReport.UpdateDataOnOpen 1}
```

The report is updated to reflect any changes made to the source data.

## **{CTRL}**

### **Syntax**

CTRL()

### **PerfectScript Syntax**

CTRL()

### **Description**

Is equivalent to using the CTRL key.



## **{DatabaseQuery}**

### **Syntax**

{DatabaseQuery *Type*; *Name*; *QueryString*; *Destination*}

### **PerfectScript Syntax**

DatabaseQuery (*Type*?:String; *Name*?:String; *QueryString*?:String; *Destination*?:String)

### **Description**

The {DatabaseQuery} macro sends the specified SQL statement to either ODBC or BDE and places the returned data in the specified block of cells.

### **Parameters**

<i>Type</i>	Type of database to query: "Paradox", "ODBC", or "BDE (Borland Database Engine)".
<i>Name</i>	Name of the database. If the type is Paradox, the name must be a path. If the type is ODBC, the name is a Data Source Name (DSN) from the user's ODBC configuration. If the name is BDE, the name is an alias name from the user's IDAPI/BDE configuration.
<i>QueryString</i>	An SQL Statement.
<i>Destination</i>	The destination block of cells where to send the result.

## **{DATE}**

### **Description**

{DATE} is equivalent to pressing Ctrl+D, which lets users enter a date or time into the active cell.

You can enter a date in a cell without using Ctrl+D. Just type a date in one of Quattro Pro's date formats--for example, 6/1/95.

### **Example**

{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.

### **► Related topics**

## {DbAlias}

### Syntax

{DbAlias WORK | PRIV, *Path*}

### PerfectScript Syntax

DbAlias (type:Enumeration {PRIV!; WORK!}; Path:String)

### Description

{DbAlias} lets you specify a private directory to hold temporary files, or a working directory where external data tables are most likely to be found.

### Parameters

WORK	WORK to specify a Working directory; PRIV to specify a Private
PRIV	directory
<i>Path</i>	path for the Working directory or the Private directory

## {DEFINE}

### Syntax

{DEFINE *Location1* <:Type1>, *Location2* <:Type2>, ...}

### Description

When you pass control to a subroutine with the `{Subroutine}` command, you can also pass arguments for use by that subroutine. If you do, 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 displays.

*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 is a valid cell 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 value resulting from a numeric formula. If it is not a numeric value, Quattro Pro treats it as a string (or string value from a formula).

### Example

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 cells 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 (`\F`) 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
```



term3 0  
amount 817.747  
payment \$817.75

txt\_area+"The monthly payment will be "&@TRIM(payment)

### **Parameters**

<i>Location</i>	Cell in which you want to store the argument being passed
<i>Type</i>	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)

### **▶ Related topics**

## **{DEL} and {DELETE}**

### **Description**

{DEL} and {DELETE} are equivalent to the Del key.

▶ **Related topics**

## **{DELETEMENU}**

### **Syntax**

{DELETEMENU *MenuPath*}

### **PerfectScript Syntax**

DeleteMenu (MenuPath:String)

### **Description**

{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 item.

### **Example**

{DELETEMENU "/File"} removes the File menu from the active menu system.

### **Parameters**

*MenuPath*      Menu in the tree to delete; type a forward slash (/) followed by the menu name; for example, to delete the Edit menu, type /Edit.

#### ► **Notes**

- You cannot delete menus between Edit and Tools on the menu bar. The area between these menu positions is reserved for context-sensitive menus that change depending on the active window. You can add menu items to menus between the Edit and Tools menus, but the new menu items will be swapped out of the menu when the context changes.
- Changes made to the menu system using this command are not saved; they are lost when you exit Quattro Pro. Each time you run a macro containing {DELETEMENU}, the menu changes appear again.
- To restore the original menu bar, use the macro command [{SETMENUBAR}](#) without an argument

#### ► **Related topics**

## {DELETEMENUITEM}

### Syntax

{DELETEMENUITEM *MenuPath*}

### PerfectScript Syntax

DeleteMenuItem (MenuPath:String)

### Description

{DELETEMENUITEM} removes the menu item specified by *MenuPath* from the menu system. Use [{DELETEMENU}](#) to remove entire menus from the active menu system.

### Example

{DELETEMENUITEM "/Edit/Clear"} removes the Clear command from the Edit menu.

{DELETEMENUITEM "/Edit/<-"} removes the first item on the Edit menu.

### Parameters

*MenuPath* Menu item in the tree to delete; enter the sequence of menu items separated by forward slashes (/); you can use <- and -> to specify an item menu at the top or bottom of a menu, respectively. For example, /File/<- specifies the first item on the File menu. You can also use numbers to identify menu items. For example, /File/0 specifies the first item on the File menu (the ID numbers start at zero).

#### ► Notes

- You can delete menu items from any menu, but if you change a context-sensitive menu (all menus between Edit and Tools on the menu bar), the change applies only to the menu in the active window. For example, suppose you use a macro to change the View menu when the notebook window is active. If you then open a chart window, the chart View menu appears--without the change. If you want the change to apply to that View menu as well, you must run the macro again.
- Changes made to the menu system using this command are not saved; they are lost when you exit Quattro Pro. Each time you run a macro containing {DELETEMENUITEM}, the menu changes appear again.
- To restore the original menu bar, use the macro command [{SETMENUBAR}](#) without an argument.

#### ► Related topics

## {DELVAR}

### Syntax

{DELVAR *VarName1* <*VarName2*,...>}

### PerfectScript Syntax

DelVar ([*VarName1*:String]; {[*VarName*:String]})

### Description

{DELVAR} deletes unused named variables. Named variables are used to control OLE objects. OLE objects are released from control at the end of macro execution, but named variables remain until you exit Quattro Pro. You can delete the unused named variables to free an object assigned to that name, and then control the object using another macro.

### Example

{DELVAR} deletes all named variables

{DELVAR calc} deletes a named variable calc

{DELVAR calc 0, calc 1, calc 3} deletes the named variables calc 0, calc 1, and calc 3.

### Parameters

*VarName* A named variable

## {DESCR}

### Syntax

{DESCR *InBlock*, *OutBlock*, *Grouped*, <*Labels*(0|1)>, <*Summary*(0|1)>, <*Largest*>, <*Smallest*>, <*Confidence*>}

### PerfectScript Syntax

DESCR (*InBlock*:String; *OutBlock*:String; [*Grouped*:String]; [*Labels*?:Enumeration {Yes!; No!}]; [*Summary*?:Enumeration {Yes!; No!}]; [*Largest*:Numeric]; [*Smallest*:Numeric]; [*Confidence*:Numeric])

### Description

{DESCR} returns a table of descriptive statistics that characterize a sample. {DESCR} is equivalent to the Descriptive Statistics analysis tool.

### Parameters

<i>InBlock</i>	One or more numeric cell values representing the input cells
<i>OutBlock</i>	Upper-left cell of the output cells
<i>Grouped</i>	"C" to group results by column or "R" to group results by row; "C" is the default
<i>Labels</i>	1 if labels are located in the first column or row of the input cells; 0 if the input cells do not contain labels; the default is 0
<i>Summary</i>	1 to display summary statistics; 0 to omit summary statistics; the default is 0
<i>Largest</i>	A value <i>n</i> which, if present, makes {DESCR} report the <i>n</i> th largest data point; if omitted, the largest data point is not reported
<i>Smallest</i>	A value <i>n</i> which, if present, makes {DESCR} report the <i>n</i> th smallest data point; if omitted, the smallest data point is not reported
<i>Confidence</i>	Confidence level of the mean; the default is 0.95

### ► Related topics

## **{DialogView}**

### **Syntax**

{DialogView *Window*}

### **PerfectScript Syntax**

DialogView (*Window*:String)

### **Description**

{DialogView} lets you edit an existing dialog box.

### **Parameters**

*Window*          Dialog window to make  
                     active

### **▶ Related topics**

## **{DialogWindow}**

### **Syntax**

{DialogWindow.*Property*}

### **Description**

{DialogWindow} is equivalent to right-clicking the title bar of a dialog window to set its properties.

{DialogWindow} commands affect the active dialog window. The next table lists the possible settings for Property. To display a property description with syntax, choose that property in the following list:

Dimension

Disabled

Grid Options

Name

Position Adjust

Title

Value

## **{DialogWindow.Dimension}**

### **Syntax**

{DialogWindow.Dimension<*Option*>}

### **PerfectScript Syntax**

DialogWindow\_Dimension\_Height (Height:Numeric)

DialogWindow\_Dimension\_Width (Width:Numeric)

DialogWindow\_Dimension\_X (XPos:Numeric)

DialogWindow\_Dimension\_Y (YPos:Numeric)

### **Description**

{DialogWindow.Dimension} 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.

### **Example**

The following macro command positions the active dialog window two pixels from the left edge of the Quattro Pro window and five pixels below the input line, and sets the width to 150 pixels and the height to 250 pixels.

```
{DialogWindow.Dimension "2,5,150,250"}
```

### **Options**

{DialogWindow.Dimension "*XPos, Ypos, Width, Height*"}

{DialogWindow.Dimension.Height *Height*}

{DialogWindow.Dimension.Width *Width*}

{DialogWindow.Dimension.X *XPos*}

{DialogWindow.Dimension.Y *YPos*}

### **▶ Related topics**



## **{DialogWindow.Disabled}**

### **Syntax**

{DialogWindow.Disabled Yes|No}

### **PerfectScript Syntax**

DialogWindow\_Disabled (Disable?:Enumeration {Yes!; No!})

### **Description**

{DialogWindow.Disabled} disables (Yes) or enables (No) the active dialog box or Toolbar. This command works only when you view a dialog box or toolbar; it does not work when you edit one.

### **▶ Related topics**

## **{DialogWindow.Grid\_Options}**

### **Syntax**

{DialogWindow.Grid\_Options *GridSize*, *ShowGrid*, *SnapToGrid*}

### **PerfectScript Syntax**

DialogWindow\_Grid\_Options (Settings:String)

### **Description**

{DialogWindow.Grid\_Options} 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 objects snap to the grid.

### **Example**

The following macro sets the distance between grid points to 10, shows the grid, and enables it.

```
{DialogWindow.Grid_Options "10, Yes, Yes"}
```

### **▶ Related topics**

## **{DialogWindow.Name}**

### **Syntax**

{DialogWindow.Name *Name*}

### **PerfectScript Syntax**

DialogWindow\_Name (Name:String)

### **Description**

{DialogWindow.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 Toolbar).

► **Related topics**

## **{DialogWindow.Position\_Adjust}**

### **Syntax**

{DialogWindow.Position\_Adjust *Depend, LeftRel, TopRel, RightRel, BottomRel, CenterHor, CenterVer*}

### **PerfectScript Syntax**

DialogWindow\_Position\_Adjust (Settings:String)

### **Description**

{DialogWindow.Position\_Adjust} specifies how the active dialog box resizes when the Quattro Pro window is resized. The arguments are equal to options in the Position Adjust dialog box.

### **▶ Related topics**

## **{DialogWindow.Title}**

### **Syntax**

{DialogWindow.Title *String*}

### **PerfectScript Syntax**

DialogWindow\_Title (Title:String)

### **Description**

{DialogWindow.Title} specifies the title that appears on the dialog box when the user is viewing it (the title does not appear when editing the dialog box).

▶ **Related topics**

## **{DialogWindow.Value}**

### **Syntax**

{DialogWindow.Value *String*}

### **PerfectScript Syntax**

DialogWindow\_Value (String:String)

### **Description**

{DialogWindow.Value} sets the initial settings of the dialog box (or Toolbar). 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.

### **Example**

The following macro command sets the initial values of a dialog box with three controls. Each setting maps to one control.

```
{DialogWindow.Value "25000,5,1st of month"}
```

### **▶ Related topics**

## {DISPATCH}

### Syntax

{DISPATCH *Location*}

### Description

{DISPATCH} is similar to {BRANCH}, except it uses *Location* differently:

- If *Location* is a cell address or one named cell, {DISPATCH} branches to the address stored in that cell.
- If *Location* is a text formula resulting in a cell or one named cell, {DISPATCH} branches to the address stored in that result address. (In Quattro Pro DOS, {DISPATCH} branches to the actual result address.)
- If *Location* contains cells or a text formula resulting in cells, {DISPATCH} branches to the first cell of those cells.

{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.

### Example

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.

```
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 current 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 current standing with P.K.  
Finance, we have decided to extend your credit limit.

Please call our office for details.

Sincerely,

James Madison (Account Officer)

### **Parameters**

*Locatio* A single cell containing the address or cell name of another macro  
*n*

▶ **Related topics**



## **{DLL}**

### **Syntax**

{DLL <DLLName.>FunctionName, Argument1, Argument2,...}

### **PerfectScript Syntax**

DLL (DLLName\_FunctionName:String; {[Argument:String]})

### **Description**

{DLL} runs a macro or returns a value from an add-in @function contained in a dynamic-link library file. The @function can have up to 16 arguments.

### **Example**

This statement calls the @function AMPLITUDE, included in the DLL Math, with two selections as arguments:

```
{DLL Math.AMPLITUDE, A1..A10, B1..B10}
```

### **Parameters**

<i>DLLName</i>	The name of a DLL file (if not already loaded)
<i>FunctionName</i>	The name of an @function contained in the DLL
<i>Argument1,Argument 2...</i>	Arguments to the @function

### **► Related topics**

## **{DLL.Load}**

### **Syntax**

{DLL.Load *DLLName*}

### **PerfectScript Syntax**

DLL\_Load (DLLName:String)

### **Description**

{DLL.Load} loads a dynamic-link library (DLL) program. You can use {DLL.Load} to load a DLL containing add-in @functions or macros. When the DLL is loaded, you can reference add-in @functions contained in the DLL without typing the DLL name. Similarly, macros contained in the DLL become resident in memory.

You can use {DLL.Load} to define a startup macro in QPW.INI.

### **Example**

{DLL.Load MYDLL} loads a DLL program named MYDLL

### **Parameters**

*DLLName*      The name of a DLL file to load

► **Related topics**

## {DODIALOG}

### Syntax

{DODIALOG *DialogName*, *OKExit?*, <*Arguments*>, <*MacUse?*>}

### Description

{DODIALOG} displays a dialog box for you or the macro to manipulate. *MacUse?* specifies how to manipulate the dialog box; 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* refers to the cells 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 (the first cell maps to the first control, the second cell maps to the second, and so on).

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.

### Parameters

<i>DialogName</i>	Name of dialog box to display
<i>e</i>	
<i>OKExit?</i>	Cell to store how the dialog box closed (1 for OK, 0 for Cancel)
<i>Arguments</i>	Cells to store the initial settings for controls in the dialog box and their final settings (optional)
<i>MacUse?</i>	1 if the user should manipulate the dialog box, 0 if the macro manipulates it (optional; 1 is the default)

### ► Related topics

## **{DOWN} and {D}**

### **Syntax**

{DOWN <Number>} or {D <Number>}

### **Description**

{DOWN} and {D} are equivalent to the Down key. The optional argument *Number* moves the selector down the corresponding number of rows. You can also use cell references or cell names as arguments.

### **Example**

{DOWN}{DOWN}	moves the selector down two rows.
{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 first cell of the named area "count".

### **Parameters**

*Number* Any positive integer (optional)

► **Related topics**

## **{DraftViewGoto}**

### **Description**

Switches from either the Objects Sheet or the sheet in Page View mode to Draft View.

### **▶ Related topics**

## **{DUPLICATE}**

### **Syntax**

{DUPLICATE *xoffset*, *yoffset* }

### **PerfectScript Syntax**

Duplicate (*xoffset*:Numeric; *yoffset*:Numeric)

### **Description**

{DUPLICATE} copies selected chart annotations or dialog box controls. If no object is selected, {DUPLICATE} does nothing.

*xoffset* and *yoffset* are measured in relative coordinates for objects in a chart window, and in pixels for dialog box objects.

### **Example**

The following macro activates a floating chart for editing, creates a rectangle in the chart, makes a copy of the rectangle, then deactivates editing.

```
{GraphEdit Chart1, 1}  
{CreateObject Rect, 147, 176, 416, 427}  
{Duplicate 269, 338}  
{GraphDeactivate}
```

### **Parameters**

<i>xoffset</i>	Offset from the left edge of <i>UpperCell</i> to the left edge of the floating object
<i>t</i>	
<i>yoffset</i>	Offset from the top edge of <i>UpperCell</i> to the top edge of the floating object
<i>t</i>	

## **{EDIT}**

### **Description**

{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.

### **▶ Related topics**

## **{EditClear}**

### **Description**

{EditClear} erases the contents and properties of the current cells, deletes selected objects from dialog and chart windows, and deletes selected floating objects. To erase cells while leaving their properties intact, use {ClearContents}.

### **▶ Related topics**



## **{EditCopy}**

### **Description**

{EditCopy} copies the selected object to the Clipboard.

▶ **Related topics**

## **{EditCut}**

### **Description**

{EditCut} removes the selected object from the spreadsheet and moves it to the Clipboard.

▶ **Related topics**

## **{EditGoto}**

### **Syntax**

{EditGoto *Block*,<*Extend?*(0|1)>}

### **PerfectScript Syntax**

EditGoto (Block:String; [Extend?:Enumeration {Yes!; No!}])

### **Description**

{EditGoto} selects and displays *Block* within spreadsheet sheets, but not the Objects sheet.

You can use {EditGoto?} or {EditGoto!} to display the Go To dialog box. {EditGoto?} lets the user manipulate the dialog box, whereas {EditGoto!} relies on the macro to manipulate it.

### **Parameters**

<i>Block</i>	Cells to display and select
<i>Extend</i>	Whether to extend the selection from the current selection to the
<i>?</i>	specified cells; 0 = no, 1 = yes; the default is 0

### **► Related topics**

## **{EditPaste}**

### **Description**

{EditPaste} copies data and its properties from the Clipboard into the notebook.

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).

### **▶ Related topics**

**{END}**

**Description**

{END} is equivalent to the End key.

▶ **Related topics**

## **{ESC} and {ESCAPE}**

### **Description**

{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}](#).

### **► Related topics**

## {Eval}

### Syntax

{Eval *Formula*}

### PerfectScript Syntax

Eval (Formula: String)

### Description

Evaluates a string as an expression, and returns the result as a string value.

### Example

"5 + 5"

Result: "10"

### Parameter

<i>Formula</i>	The string to evaluate
----------------	------------------------

## {EXEC}

### Syntax

{EXEC *AppName*, *WindowMode*, <*ResultLoc*>}

### Description

{EXEC} lets you run other Windows applications or DOS commands. *AppName* can be any valid command string you could type in the Windows Run dialog box. *AppName* can contain the path of the application. If no path is given, 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 an instance number.

### Example

{EXEC "C:\COREL\SUITE8\PROGRAMS\WPWIN8.EXE",2} runs Corel WordPerfect for Windows as a minimized application

{EXEC "NOTEPAD.EXE",1} runs the Windows Notepad and displays it just as if it were run from the Start button on the taskbar.

{EXEC "C:\COMMAND.COM /C DIR>TEST.TXT",1} stores the current directory listing in the file TEST.TXT for Windows, except for Windows NT 4.0.

{EXEC "C:\CMD.EXE /C DIR>TEST.TXT",1} stores the current directory listing in the file TEST.TXT in Windows NT 4.0.

### Parameters

<i>AppName</i>	Name (in quotes) of the application to run (up to 100 characters)
<i>WindowMode</i>	Size state of the application's window: 1 or 101 for normal size, 2 or 102 for minimized, 3 or 103 for maximized; use 101, 102, 103 to suspend macro execution until the application terminates
<i>ResultLoc</i>	Cell containing the coded explanation of the operation's success

### ► Related topics

## {EXECAUTO}

### Syntax

```
{EXECAUTO AutoExpr1 <,> AutoExpr2 >
```

### PerfectScript Syntax

```
ExecAuto (AutoExpr1:String; {[AutoExpr:String]})
```

### Description

{EXECAUTO} executes one or more methods in another application, but drops any return values.

### Example

{EXECAUTO calc.Display()} asks DispCalc to display its current input value.

{EXECAUTO calc.Button(A1), calc.Display()} passes the value in A1 as an input to DispCalc and asks DispCalc to display it.

### Parameters

*AutoExpr1,2*, .. One or more automation expressions

## {ExecMacro}

### Syntax

```
{ExecMacro <Filename; > Macro}
```

### PerfectScript Syntax

```
ExecMacro ([Filename: String;] Macro: String)
```

### Description

Starts Quattro Pro, opens the file, runs the macro, and exits Quattro Pro.

### Parameters

<i>Filename</i> [optional]	The name of the file that contains the macro you want to run.
<i>Macro</i>	The name of the macro you want to run.

## {EXECUTE}

### Syntax

```
{EXECUTE DDEChannel, Macro, <ResultLoc>}
```

### Description

With {EXECUTE} you can make other applications that support DDE run their macros. The macro to run is stored in the string *Macro*. It must be in the syntax the application normally uses (refer to the documentation for each application). 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 {REQUEST} and {POKE} for more information on receiving data from and sending data to other applications using DDE.

### Example

The following example initiates a DDE conversation with Corel WordPerfect's macro server and opens REPORT.DOC in Corel WordPerfect.

```
Channel 1  
{INITIATE "WPwin7_Macros", "COMMANDS", channel}  
{EXECUTE channel, "FileOpen ("\"report.doc\"")" }
```



{TERMINATE channel}

### **Parameters**

<i>DDEChanne</i>	Channel ID number of the application to run a macro in
<i>l</i>	
<i>Macro</i>	String containing the macro command(s) to run
<i>ResultLoc</i>	A cell indicating the result of the operation

### **▶ Related topics**

## **{EXPON}**

### **Syntax**

{EXPON *InBlock*,*OutBlock*,<*Damping*>,<*StdErrs*>}

### **PerfectScript Syntax**

EXPON (InBlock:String; OutBlock:String; [Damping:Numeric]; [StdErrs?:Enumeration {Yes!; No!}])

### **Description**

{EXPON} performs exponential smoothing on a series of values. {EXPON} is equivalent to the Exponential Smoothing analysis tool.

### **Parameters**

<i>InBlock</i>	Input cells containing a single column or row with at least four numeric values; the cells must not contain labels
<i>OutBlock</i>	Upper-left cell of the output cells
<i>Damping</i>	Damping factor used as the exponential smoothing constant; indicates the percentage for error to adjust each prior forecast value; must be $\geq 0$ ; the default is 0.3
<i>StdErrs</i>	Flag indicating whether standard errors are included in the output table: yes (1) or no (0); the default is 0

### **▶ Related topics**

# {ExportGraphic}

## Syntax

{ExportGraphic *Filename*,<*GrayScale?*(0|1)>,<*Compression?*(0|1)>}

## PerfectScript Syntax

ExportGraphic (Filename:String; [GrayScale?:Enumeration {Yes!; No!}]; [Compression?:Enumeration {Yes!; No!}])

## Description

{ExportGraphic} saves selected graphic objects to one of several file types with optional gray-scaling and compression.

You can use {ExportGraphic?} or {ExportGraphic!} to display the Export Graphics File dialog box.

{ExportGraphic?} lets the user manipulate the dialog box, whereas {ExportGraphic!} relies on the macro to manipulate it.

## Parameters

<i>Filename</i>	Name of the graphic file to export
<i>GrayScale?</i>	Whether to gray-scale: no (0), yes (1); the default is 0
<i>Compression</i>	Type of .TIF file compression to use: none (0) or PackBits (1);
<i>?</i>	the default is 0

## **{FileClose} and {FileCloseAll}**

### **Syntax**

```
{FileClose <DoSave? (0|1)>}  
{FileCloseAll <DoSave? (0|1)>}
```

### **PerfectScript Syntax**

```
FileClose ([DoSave?:Enumeration {Yes!; No!}])  
FileCloseAll ([DoSave?:Enumeration {Yes!; No!}])
```

### **Description**

{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.

### **Options**

{FileClose <DoSave? (0 1)>}	Closes all views of the active notebook
{FileCloseAll <DoSave? (0 1)>}	Closes all open notebooks

### **► Related topics**

## {FileCombine}

### Syntax

{FileCombine *Filename*, <*Blocks*>, Add | Subtract | Multiply | Divide | Copy}

### PerfectScript Syntax

FileCombine (FileName:String; [Blocks:String]; Operation:Enumeration {Copy!; Add!; Subtract!; Multiply!; Divide!})

### Description

{FileCombine} lets you copy all or part of a notebook into any area of the active notebook. 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.

You can use {FileCombine?} or {FileCombine!} to display the Combine Files dialog box. {FileCombine?} lets you manipulate the dialog box, whereas {FileCombine!} relies on the macro to manipulate it.

### Parameters

<i>Filename</i>	Name of the file to combine
<i>Blocks</i>	Selection or selections within <i>Filename</i> to combine (optional)

#### ► Note

- This command is obsolete.

## **{FileExit}**

### **Syntax**

{FileExit <DoSave?(0|1)>}

### **PerfectScript Syntax**

FileExit ([DoSave?:Enumeration {Yes!; No!}])

### **Description**

{FileExit} 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.

### **Parameters**

<i>DoSave?</i>	Whether to display a save prompt for modified files: no (0), yes (1); 1 is the default
----------------	---

### **▶ Related topics**

## {FileExtract}

### Syntax

{FileExtract Formulas | Values, *Blocks*, *Filename*, <Replace | Backup | Confirm>}

### PerfectScript Syntax

FileExtract (What:Enumeration {Formulas!; Values!}; Blocks:String; Filename:String; [Option:Enumeration {Confirm!; Replace!; Backup!}])

### Description

{FileExtract} 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. The optional argument--"Replace," "Backup," or "Confirm"--indicates how to treat an existing file with the same name (without displaying a prompt).

You can use {FileExtract?} or {FileExtract!} to display the Extract To File dialog box. {FileExtract?} lets you manipulate the dialog box, whereas {FileExtract!} relies on the macro to manipulate it.

### Parameters

<i>Blocks</i>	Selection or selections to extract
<i>Filename</i>	Name of the new file containing Blocks

## {FileImport}

### Syntax

{FileImport "*Filename*","ASCII Text File"|"Comma and "" Delimited File"|"Only Commas"|"Parse Expert"}

### PerfectScript Syntax

FileImport (Filename:String; Method:String)

### Description

{FileImport} copies a text file into the active sheet of a notebook. Enter the option string that describes the type of file to import.

You can use {FileImport?} or {FileImport!} to display the Text Import dialog box. {FileImport?} lets you manipulate the dialog box, whereas {FileImport!} relies on the macro to manipulate it.

### ► Related topics

## {FileNew}

### Syntax

{FileNew <TemplateName>}

### PerfectScript Syntax

FileNew ([TemplateName:String])

### Description

{FileNew} opens a blank notebook or a notebook based on a QuickTemplate.

You can use {FileNew?} or {FileNew!} to display the New File dialog box. {FileNew?} lets you manipulate the dialog box, whereas {FileNew!} relies on the macro to manipulate it.

FileNew opens only templates in the default QuickTemplates folder.

### Example

The following macro opens a blank notebook:

```
{FileNew}
```

The following macro opens a new notebook based on the 7-Year Balloon Loan QuickTemplate:

```
{FileNew "7 Year Balloon Loan"}
```

### Parameters

<i>TemplateName</i>	The name of a QuickTemplate
---------------------	-----------------------------

## {FileOpen}

### Syntax

{FileOpen *Filename*,<"Open Supporting"|"Update References"|"None ", *Open as Copy* 0|1>}

### PerfectScript Syntax

FileOpen (Filename:String; [Option:Enumeration {Open!; Update!; None!}])

### Description

{FileOpen} opens the specified file.

You can use {FileOpen?} or {FileOpen!} to display the Open File dialog box. {FileOpen?} lets you manipulate the dialog box, whereas {FileOpen!} relies on the macro to manipulate it.

### Parameters

<i>Filename</i>	Name of the file to open.
<i>Open as Copy</i>	Yes or No, (1 or 0), 0 is the default.

#### ► Note

- To open two or more URL files using Quattro Pro Native macros script, insert a {WAIT} statement after each {FileOpen} statement.



## {FileRetrieve}

### Syntax

{FileRetrieve *Filename*, <"Open Supporting" | "Update References" | "None ">}

### PerfectScript Syntax

FileRetrieve (Filename:String; [Option:Enumeration {Open!; Update!; None!}])

### Description

{FileRetrieve} loads a notebook into the active notebook, replacing any existing data there.

You can use {FileRetrieve?} or {FileRetrieve!} to display the Retrieve File dialog box. {FileRetrieve?} lets you manipulate the dialog box, whereas {FileRetrieve!} relies on the macro to manipulate it.

### Parameters

<i>Filename</i>	Name of the file to retrieve
-----------------	------------------------------

## {FileSave}, {FileSaveAll}, and {FileSaveAs}

### Syntax

{FileSave <Replace|Backup|Confirm>}

{FileSaveAll <Replace|Backup|Confirm>}

{FileSaveAs *Filename*, <Replace | Backup | Confirm>,, <*FileType*>}

### PerfectScript Syntax

FileSave ([Option:Enumeration {Confirm!; Replace!; Backup!}])

FileSaveAll ([Mode:Enumeration {Confirm!; Replace!; Backup!}])

FileSaveAs (Filename:String; [Option:Enumeration {Confirm!; Replace!; Backup!}]; [reserved:Numeric]; [FileType:String])

### Description

{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).

The optional <*FileType*> argument for {FileSaveAs} specifies the type of file to save and is equivalent to the Save File As Type option in the Save File dialog box. If you do not specify a file type, the default is "QPW v6."

You can use {FileSaveAs?} or {FileSaveAs!} to display the Save File dialog box. {FileSaveAs?} lets you manipulate the dialog box, whereas {FileSaveAs!} relies on the macro to manipulate it.

The following table shows the available file types. For file types with abbreviated names, a short description is provided.

File Types	Description
QPW v9/v10	Quattro Pro for Windows, version 9.0 and 10
QPW v7/v8	Quattro Pro for Windows, version 7.0 and 8.0
QPW v6	Quattro Pro for Windows, version 6.0
QPW	Quattro Pro for Windows, version 1.0 and 5.0
QP/DOS	Quattro Pro for DOS
Excel v5/v7	Excel, Version 5.0 and Version 7.0
Excel	Excel, Version 4.0
1-2-3 v4/v5	1-2-3, Version 4 and Version 5
1-2-3 v3.x	1-2-3, Version 3x
1-2-3 v2.x	1-2-3, Version 2x
1-2-3 v1.0	1-2-3, Version 1.0
1-2-3 Ed.	1-2-3, Educational Version
Paradox	

dBASE IV	
dBASE III	
dBASE II	
Text	tab-delimited text
DIF	VisiCalc
SYLK	Multiplan
HTML	Hypertext Markup Language files, Version 3 (for distribution on the Internet's World Wide Web)

## Example

To close all files and save without confirmation, use this macro:

```
{FileSaveAll Replace}
```

```
{FileCloseAll 0}
```

## Options

{FileSave <Replace  Backup Confirm>}	Saves the notebook to the name under which you last saved it
{FileSaveAll <Replace  Backup Confirm>}	Saves the file over a previous version with the same name
{FileSaveAs <i>Filename</i> , <Replace   Backup   Confirm>,, < <i>FileType</i> >}	Saves the notebook under a new name you specify

## ► Related topics

## {FileSend}

### Syntax

```
{FileSend <Filename>}
```

### PerfectScript Syntax

```
FileSend ([Filename:String])
```

### Description

{FileSend} lets you send notebook sheets via one of your mail systems.

### Example

{FileSend MYSTATUS.WB3} sends the notebook MYSTATUS.WB3 to another user.

### Options

<pre>{FileSend &lt;Filename&gt; }</pre>	Sends selected text or an entire notebook by e-mail
---	---

## {FILESIZE}

### Syntax

```
{FILESIZE Location}
```

### Description

{FILESIZE} calculates the number of bytes in a file previously opened using [{OPEN}](#) and places the result in *Location* as a value. If *Location* refers to cells, the result is placed in the upper-left cell of the cells. 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.

### Example

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
```

### Parameters

<i>Location</i>	Any cell address or cell name
-----------------	-------------------------------

#### ► Note

- This command is obsolete.

## **{ FileVersion.Retrieve }**

### **Syntax**

{FileVersion.Retrieve *Filename*}

### **PerfectScript Syntax**

FileVersion\_Retrieve (Filename?:String)

### **Description**

{FileVersion.Retrieve } retrieves any archived version of a file

## **{ FileVersion.Retrieve\_Current }**

### **Syntax**

{FileVersion.Retrieve\_Current}

### **PerfectScript Syntax**

FileVersion\_Retrieve\_Current ()

### **Description**

{FileVersion.Retrieve\_Current} retrieves the most current version of the file.

## **{ FileVersionSave }**

### **Syntax**

{FileVersionSave}

### **PerfectScript Syntax**

FileVersionSave ()

### **Description**

{FileVersionSave} saves the current file as a different version.

## {FLOATCOPY}

### Syntax

{FLOATCOPY *UpperCell*, *xoffset*, *yoffset*}

### PerfectScript Syntax

FloatCopy (UpperCell:String; xoffset:Numeric; yoffset:Numeric)

### Description

{FLOATCOPY} lets you copy a floating object in the active notebook window. The item to copy is selected using [{SELECTFLOAT}](#). The new position in {FLOATCOPY} is specified as a positive offset from a cell in the notebook.

To copy a floating chart to another notebook, specify a notebook as well as a cell for *UpperCell*.

### Example

The following macro selects the floating chart Inserted1 and copies it to [SALES]A:C10.

```
{SELECTFLOAT Inserted1}
```

```
{FLOATCOPY [SALES]A:C10,0,0}
```

### Parameters

<i>UpperCell</i>	Cell containing the new upper-left corner of the floating object
<i>xoffset</i>	Offset in twips from the left edge of <i>UpperCell</i> to the left edge of the floating object
<i>yoffset</i>	Offset in twips from the top edge of <i>UpperCell</i> to the top edge of the floating object

### ► [Related topics](#)

# {FLOATCREATE}

## Syntax

{FLOATCREATE *Type*, *UpperCell*, *xoffset*, *yoffset*, *LowerCell*, *xoffset2*, *yoffset2*, <*Text*>|*StartCorner*}

## PerfectScript Syntax

FloatCreate (Type:String; UpperCell:String; xoffset:Numeric; yoffset:Numeric; LowerCell:String; xoffset2:Numeric; yoffset2:Numeric; [TextOrStartCorner:Any])

## Description

{FLOATCREATE} lets you create macro buttons, floating charts, or a draw layer objects (lines, arrows, rectangles, rounded rectangles, ellipses, or text boxes) in the active notebook window. Use [{CREATEOBJECT}](#) to create objects in dialog windows or chart windows.

All positions in {FLOATCREATE} are positive offsets from cells in the notebook containing the upper-left and lower-right corners of the object.

### ► Notes

- If you need to modify the floating object after creating it, change the property settings immediately after creation. It is selected then, so you will not need to click it or use [{SELECTFLOAT}](#).
- You should also change the name at this time and document it for later use with {SELECTFLOAT}.

## Example

The following macro creates a macro button that covers the cells A1..B2, then stores the name of the button in A26. The button reads Save File:

```
{FLOATCREATE Button,A1,0,0,C3,0,0, "Save File"}  
{GETPROPERTY A26, "Object_Name"}
```

The following macro creates a button 50 twips to the right and 50 twips below the upper-left corner of the button in the previous example. It reads Open File:

```
{FLOATCREATE Button,A1,50,50,C3,50,50, "Open File"}
```

The following macro creates a floating chart that is offset 35 twips from the cells C2..E10, but the same size:

```
{GraphNew Chart3}  
{FLOATCREATE Chart,C2,35,35,E10,35,35,"Chart3"}
```

The following macro creates a floating arrow over the cells B8..D11. The arrow starts at the southwest corner of the cells, and ends with an arrowhead at the northwest corner.

```
{FloatCreate Arrow,A:B8,0,0,A:D11,945,45,4}
```

The following macro creates a floating ellipse over the cells E10..E13, then fills the ellipse with a red color.

```
{FloatCreate Ellipse,A:E10,0,120,A:E13,945,240}  
{Setproperty Fill_Color, "255,0,0"}
```

## Parameters

<i>Type</i>	Floating object to create: Chart, Button, Line, Arrow, Rect, Rounded_Rect, Ellipse, or Text
<i>UpperCell</i>	Cell containing the upper-left corner of the chart or macro button
<i>xoffset</i>	Offset in twips from the left edge of <i>UpperCell</i> to the left edge of the floating object
<i>yoffset</i>	Offset in twips from the top edge of <i>UpperCell</i> to the top edge of the floating object
<i>LowerCell</i>	Cell containing the lower-right corner of the chart or macro button
<i>xoffset2</i>	Offset in twips from the left edge of <i>LowerCell</i> to the right Edge of the floating object
<i>yoffset2</i>	Offset in twips from the top edge of <i>LowerCell</i> to the bottom edge of the floating object
<i>Text</i>	For Chart, the named chart to display; for Button, the button text
<i>StartCorner</i>	For Line or Arrow, a number representing the starting corner; 1 = northwest, 2 = northeast, 3 = southeast, 4 = southwest (for example, an arrow pointing up and to the right would have a <i>StartCorner</i> of 4) You must this value within quotations (""). ""



► **Related topics**

## {FLOATMOVE}

### Syntax

{FLOATMOVE *UpperCell*,*xoffset*,*yoffset*}

### PerfectScript Syntax

FloatMove (*UpperCell*:String; *xoffset*:Numeric; *yoffset*:Numeric)

### Description

{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.

To move a floating chart to another notebook, specify a notebook as well as a cell for *UpperCell*.

### Example

The following macro selects the floating chart *Inserted1* and moves it to [SALES]A:C10.

```
{SELECTFLOAT Inserted1}
```

```
{FLOATMOVE [SALES]A:C10,0,0}
```

### Parameters

<i>UpperCell</i>	Cell containing the new upper-left corner of the floating object
<i>xoffset</i>	Offset in twips from the left edge of <i>UpperCell</i> to the left edge of the floating object
<i>yoffset</i>	Offset in twips from the top edge of <i>UpperCell</i> to the top edge of the floating object

### ► [Related topics](#)

## {FloatOrder}

### Syntax

{FloatOrder.Option}

### PerfectScript Syntax

FloatOrder\_Backward ()

FloatOrder\_Forward ()

FloatOrder\_ToBack ()

FloatOrder\_ToFront ()

### Description

{FloatOrder} works on selected objects to arrange layers of floating charts and other floating objects in the notebook window.

### Options

{FloatOrder.ToBack}	Send the selected object to the back layer
{FloatOrder.Backward}	Send the selected object back one layer
{FloatOrder.ToFront}	Send the selected object to the front layer
{FloatOrder.Forward}	Send the selected object forward one layer

## {FLOATSIZE}

### Syntax

{FLOATSIZE UpperCell,xoffset,yoffset,LowerCell,xoffset2,yoffset2}

### PerfectScript Syntax

FloatSize (UpperCell:String; xoffset:Numeric; yoffset:Numeric; LowerCell:String; xoffset2:Numeric; yoffset2:Numeric)

### Description

{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.

### Parameters

<i>UpperCell</i>	Cell containing the new upper-left corner of the chart or macro button
<i>xoffset</i>	Offset in twips from the left edge of UpperCell to the left edge of the floating object
<i>yoffset</i>	Offset in twips from the top edge of UpperCell to the top edge of the floating object
<i>LowerCell</i>	Cell containing the new lower-right corner of the chart or macro button
<i>xoffset2</i>	Offset in twips from the left edge of LowerCell to the right edge of the floating object
<i>yoffset2</i>	Offset in twips from the top edge of LowerCell to the bottom edge of the floating object

### ► [Related topics](#)

## **{FLOATTEXT}**

### **Syntax**

{FLOATTEXT *String*}

### **PerfectScript Syntax**

FloatText (String:String)

### **Description**

{FLOATTEXT} replaces the text in the selected text box with the specified string. The text box can be on a notebook sheet or in a chart window.

### **Example**

The following macro selects a floating text box named Text1 and replaces the text in it with "Quarterly Sales Report."

```
{SELECTFLOAT Text1}  
{FLOATTEXT "Quarterly Sales Report"}
```

### **Parameters**

*String*                      String of characters used to replace text in the text box

### **▶ Related topics**

## {FOR}

### Syntax

{FOR *CounterLoc*,*Start#*,*Stop#*,*Step#*,*StartLoc*}

### Description

{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.

### Example

{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, 1994,1997,1,_list}  
{; return to original position}  
{LEFT}{END}{LEFT}
```

```
counter_list{PUTCELL +counter}  
{DOWN}  
PUTCELL "Qtr1"{RIGHT}  
{PUTCELL "Qtr2"{RIGHT}  
{PUTCELL "Qtr3"{RIGHT}  
{PUTCELL "Qtr4"{RIGHT 2}{UP}
```

### Parameters

<i>CounterLo</i> <i>c</i>	Cell used to track the number of macro iterations
<i>Start#</i>	Initial value to place in CounterLoc
<i>Stop#</i>	Maximum value for CounterLoc
<i>Step#</i>	Amount added to CounterLoc after each iteration
<i>StartLoc</i>	Cell containing the subroutine to run

#### ► Note

- This command is obsolete.

#### ► Related topics

## **{FORBREAK}**

### **Description**

{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.

### **Example**

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
```

#### **► Note**

- This command is obsolete.

#### **► Related topics**

## **{Form}**

### **Syntax**

{Form <Block>}

### **PerfectScript Syntax**

Form(Block?:<Block>)

### **Description**

Equivalent to Tools ▶ Database Tools

▶ Form. Lets you create forms for entering and finding data records without programming.

### **Parameter**

*Block*

A database block including field labels and records

▶ **Related topics**

## **{FOURIER}**

### **Syntax**

{FOURIER *InBlock*,*OutBlock*,<*Inverse*>}

### **PerfectScript Syntax**

FOURIER (*InBlock*:String; *OutBlock*:String; [*Inverse*?:Enumeration {Yes!; No!})

### **Description**

{FOURIER} performs a fast Fourier transformation on cells of data. {FOURIER} is equivalent to the Fourier analysis tool.

### **Parameters**

<i>InBlock</i>	One or more numeric cell values representing the input cells; can be real or complex numbers; the number of values in <i>InBlock</i> must be a power of 2 between 2 and 1024 inclusive (for example, 2, 4, 8, 16,...); if the number of values in <i>InBlock</i> does not equal a power of 2, pad the cells with additional zeros
<i>OutBlock</i>	Upper-left cell of the output cells
<i>Inverse</i>	0 to perform a Fourier transformation; 1 to perform the inverse Fourier transformation; the default is 0

### **► Related topics**



## {Frequency}

### Syntax

{Frequency.Option}

### PerfectScript Syntax

Frequency\_Bin\_Block (Block:String)

Frequency\_Go ()

Frequency\_Reset ()

Frequency\_Value\_Block (Block:String)

### Description

{Frequency} 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.

You can use {Frequency?} or {Frequency!} to display the Frequency Tables dialog box. {Frequency?} lets you manipulate the dialog box, whereas {Frequency!} relies on the macro to manipulate it.

### Example

The following macro counts the data in cells C1..E13 of sheet 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}
```

### Options

{Frequency.Bin_Block <i>Block</i> }	Specifies cells that define value intervals or "bins" of values to be counted
{Frequency.Go}	Accepts the frequency settings
{Frequency.Reset}	Clears all settings
{Frequency.Value_Block <i>k Block</i> }	Specifies the cells or list of cells containing values to be counted

## {FTESTV}

### Syntax

{FTESTV *InBlock1*,*InBlock2*,*OutBlock*,<*Labels*>}

### PerfectScript Syntax

FTESTV (*InBlock1*:String; *InBlock2*:String; *OutBlock*:String; [*Labels*?:Enumeration {Yes!; No!}]

### Description

{FTESTV} performs a two-sample F-test to compare population variances. {FTESTV} is equivalent to the F-Test analysis tool.

### Parameters

<i>InBlock1</i>	The first input cells containing a column or row of numeric values
<i>InBlock2</i>	The second input cells containing a column or row of numeric values
<i>OutBlock</i>	Upper-left cell of the output cells
<i>Labels</i>	1 if labels are located in the first column or row of the input cells; 0 if the input cells do not contain labels; the default is 0

### ► Related topics

## **{FUNCTIONS}**

### **Description**

{FUNCTIONS} is equivalent to the Functions key Alt+F3, which displays a list of @functions to enter in the input line.

### **▶ Related topics**



## **{GET}**

### **Syntax**

{GET *Location*}

### **Description**

{GET} pauses macro execution, accepts one keystroke (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 is not passed to Quattro Pro.

{GET} is useful for creating macros that run in the background while you work. It can detect each key, and restrict the actions you perform. You can use {LOOK} with {GET} to check the typeahead buffer for user response.

### **Example**

The following example asks you if you want to continue.

```
\F{RIGHT 10}
{; display msg_area in top left of screen}
{GOTO}msg_area~
```

```
_again      Press Y to continue...~
```

```
{GET keystroke}
```

```
{IF keystroke<>"Y"}{BEEP2}{HOME}{QUIT}
```

```
{BEEP 4}{BRANCH _again}
```

```
keystroke
```

```
msg_area Press Y to continue...
```

### **Parameters**

*Location*      Cell in which to store the keystroke you entered.

### **▶ Related topics**

## **{GetCellFormula}**

### **Syntax**

{GetCellFormula *Cell*}

### **PerfectScript Syntax**

GetCellFormula (Cell: String)

### **Description**

Returns the unparsed form of a referenced formula. If the cell is a number, it will return the numeric text. If the cell is a label, it will be prefixed by the prefix char (' , " , or ^). If the cell is a formula, it will return the formula itself.

### **Parameter**

*Cell*            The cell

### **▶ Related topics**

## **{GetCellValue}**

### **Syntax**

{GetCellValue *Cell*}

### **PerfectScript Syntax**

GetCellValue (Cell: String)

### **Description**

Retrieves the cell contents as it is displayed, not as its value. If the cell contains a formula, it returns the result of the formula, including its numeric format.

### **Parameter**

*Cell*            The cell

### **▶ Related topics**

# {GETDIRECTORYCONTENTS}

## Syntax

{GETDIRECTORYCONTENTS *Block*,<*Path*>}

## PerfectScript Syntax

GetDirectoryContents (Block:String; [Path:String])

## Description

{GETDIRECTORYCONTENTS} enters an alphabetized list of file names (determined by the path and DOS wildcard specified by *Path*) into *Block*; if *Path* is not included, {GETDIRECTORYCONTENTS} lists all the files in the current directory. *Path* must contain a DOS wildcard like \*.BAT or \*.\*.

## Example

{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:\COREL\SUITE8\\*.\*"} fills the cells A2..C7 with a list of the files in the Quattro Pro 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 cells are only filled with the first 18.

{GETDIRECTORYCONTENTS C7,"C:\COREL\SUITE8\SAMPLES\\*.W??"} fills column C (starting at row 7) with a list of the files in the COREL\SUITE8\SAMPLES directory on drive C that have file extensions beginning with W.

## Parameters

<i>Block</i>	Cells to enter list of files into
<i>Path</i>	Path and wildcard specifying the list (optional)

### ► Note

- If *Block* is one cell, {GETDIRECTORYCONTENTS} overwrites any information beneath the cell (if it finds more than one file). To restrict the file names to specific cells, set *Block* to more than one cell.

### ► Related topics

## **{GETLABEL}**

### **Syntax**

`{GETLABEL Prompt,Location}`

### **Description**

{GETLABEL} pauses macro execution, displays *Prompt* in a dialog box, and accepts keystrokes until you choose OK or press Enter. At that time, Quattro Pro stores the characters typed as a label in *Location*. Unlike {GET}, it does not 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.

If {PANELOFF} is in effect, {GETLABEL} ignores it, letting you view and update the menus, status line, and input line as you type. Once input is completed (indicated by Enter or choosing OK), then these portions of the display are turned off again.

### **Example**

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 your last name in the cell named last\_cell:

```
\F{GETLABEL "Enter your last name:",last_cell}
```

```
last_cell
```

### **Parameters**

*Prompt* String displayed to you as a prompt  
*Locatio* Cell in which to store your response; when {GETLABEL} is used in  
*n* OLE automation, *Location* can also be a named variable

### **▶ Related topics**



## **{GETNUMBER}**

### **Syntax**

{GETNUMBER *Prompt,Location*}

### **Description**

{GETNUMBER} is like [{GETLABEL}](#), but accepts only a numeric value, a formula resulting in a numeric value, or the cell address or cell 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.

### **Example**

The following example stores your age in the cell named age\_cell. If you do not enter a number, 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
```

### **Parameters**

<i>Prompt</i>	String displayed to you as a prompt
<i>Location</i>	Cell in which to store the user's response; when {GETNUMBER} is used in OLE automation, <i>Location</i> can also be a named variable

### **▶ Related topics**

## {GetObjectPageContents}

### Syntax

{GetObjectPageContents Block<;ObjectType>}

### PerfectScript Syntax

GetObjectPageContents(Block?: Range, Objects?: ObjectType)

### Description

{GetObjectPageContents} stores a list of the objects contained on the Object Page on the Quattro Pro desktop in Block. Objects are charts, dialogs, maps, and slideshows.

### Parameters

<i>Block</i>	Cells in which to store object names
<i>ObjectType</i>	All (default), Dialog, Chart, Map, and SlideShow.

#### ► Note

- If Block is one cell, {GetObjectPageContents} overwrites any information beneath the cell if it finds more than one open window. To restrict the window names to specific cells, set Block to more than one cell.

## **{GETOBJECTPROPERTY}**

### **Syntax**

{GETOBJECTPROPERTY *Cell, Object.Property*}

### **PerfectScript Syntax**

GetObjectProperty (Cell:String; ObjectProperty:String)

### **Description**

{GETOBJECTPROPERTY} lets you view objects in Quattro Pro without using the mouse, including objects normally not selectable (like the application title bar). You can also study selectable objects, such as blocks and annotations, with {GETPROPERTY}. See [{SETOBJECTPROPERTY}](#) for the syntax of *Object.Property*.

### **Example**

{GETOBJECTPROPERTY A23, "Active\_Notebook.Zoom\_Factor"} stores the Zoom Factor property's current setting in cell A23.

{GETOBJECTPROPERTY B42, "/File/Exit.Enabled"} stores whether Exit is operational or not in the cell B42.

### **Parameters**

<i>Cell</i>	Cell in which to store the property setting
<i>Object</i>	Name of the object to study
<i>Property</i>	Property of the object to study

### **► [Related topics](#)**

## {GETPOS}

### Syntax

{GETPOS *Location*}

### Description

{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 is 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.

### Example

The following example opens the text file TEST.TXT, reads in a line, and calculates the length of the line by subtracting the starting position from the ending position and 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}
```

```
inputThis is a short line.
```

```
start0
end 22
num_char 21
```

### Parameters

*Location*      Cell in which to store the retrieved value

### ► Related topics

## **{GETPROPERTY}**

### **Syntax**

{GETPROPERTY *Cell*,*Property*}

### **PerfectScript Syntax**

GetProperty (Cell:String; PropertyName:String)

### **Description**

{GETPROPERTY} lets you study the property settings of whatever object is selected. *Property* is the property to view its setting is stored in *Cell*. (see [Property Reference](#) for a list of properties)

### **Example**

{GETPROPERTY A23,"Text\_Color"} stores the Text Color setting of the selected object in the cell A23.

{GETPROPERTY B42,"Box\_Type"} stores the border style of the selected object in cell B42.

### **Parameters**

*Cell*                    Cell in which to store the property setting

*Property*              Property of the selected object to study

► [Related topics](#)

## **{GETWINDOWLIST}**

### **Syntax**

{GETWINDOWLIST *Block*}

### **PerfectScript Syntax**

GetWindowList (Block:String)

### **Description**

{GETWINDOWLIST} stores a list of the windows open on the Quattro Pro desktop in *Block*, including dialog windows and chart windows. Windows currently hidden are not included.

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 specific cells, set *Block* to more than one cell.

### **Example**

{GETWINDOWLIST A2..C5} stores a list of open windows in the cells 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 cells.

### **Parameters**

*Block*            Cells to store window names in

### **▶ Related topics**

## **{GOTO}**

### **Description**

{GOTO} lets you move the selector to a specific location. Note that you are unable to manipulate the "Enter address to go to" dialog box using the {GOTO} macro. A related macro, {QGOTO},mac\_QGOTO selects the cell or cells specified as its destination, while {GOTO} moves to the cell in the upper-left corner of the specified cells.

This command is for compatibility with Quattro Pro for DOS.

### **▶ Related topics**

## **{GRAPH}**

### **Description**

{Graph} is equivalent to the Chart key, F11, which displays the current chart.

#### **▶ Note**

- This command is obsolete.

#### **▶ Related topics**



## {GRAPHCHAR}

### Syntax

{GRAPHCHAR *Location*}

### Description

{GRAPHCHAR} returns the key you press to leave a chart or to remove a message box. The character is stored at *Location*. This command lets you create a chart or message that functions as a menu of choices. You can use the character returned to branch or display any other data.

{GRAPHCHAR} works with {MESSAGE} to record what key you pressed to remove a message box. It captures only characters you can normally type into a cell; special characters such as Esc are not stored.

### Example

The following example displays a message box (its contents being the\_msg) and returns the character you pressed to remove it. The key pressed is stored in cells named the\_key.

```
\A {MESSAGE the_msg,15,15,0}
```

```
{GRAPHCHAR the_key}
```

```
the_msg Press C to continue or any other key to quit...
```

The next example displays a chart and waits for you to press a key. This macro begins with the command found next to "\_graphic" and assumes that the current file contains charts by the name of "line," "pie," "bar," and "area." If you press P, B, or A, Quattro Pro displays a (predefined) pie chart, bar chart, or area chart, respectively. The macro stops playing when you press any key that is not P, B, or A.

```
the_chart bar
```

```
the_key b
```

```
_graphic {GraphView "line"}
```

```
_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_chart}
```

```
{GraphView the_chart}
```

```
{BRANCH _continue}
```

### Parameters

<i>Location</i>	Cell address or the cell name where the returned character should be stored
<i>n</i>	

#### ► Note

- This command is obsolete.

#### ► Related topics

## {GraphCopy}

### Syntax

```
{GraphCopy FromChart, DestChart, <Style?(0|1)>, <Data?(0|1)>, <Annotations?(0|1)>}
```

### PerfectScript Syntax

```
GraphCopy (FromGraph:String; DestGraph:String; [Style?:Enumeration {Yes!; No!}]; [Data?:Enumeration {Yes!; No!}]; [Annotations?:Enumeration {Yes!; No!}])
```

### Description

{GraphCopy} copies the style, data, and/or annotation objects from one chart to another (within a notebook or between notebooks).

You can use {GraphCopy?} or {GraphCopy!} to display the Paste Special Chart dialog box. {GraphCopy?} lets you manipulate the dialog box, whereas {GraphCopy!} relies on the macro to manipulate it.

### Parameters

<i>FromChart</i>	Chart containing the style, data, or annotation objects to copy
<i>DestChart</i>	New chart (the copy)
<i>Style?</i>	Whether to copy properties that affect the appearance of the chart: yes (1), no (0)
<i>Data?</i>	Whether to copy chart data: yes (1), no (0)
<i>Annotations?</i>	Whether to copy annotation objects: yes (1), no (0)

## {GraphDeactivate}

### Description

{GraphDeactivate} deactivates a floating chart that has been activated for editing.

### Example

The following macro activates a floating chart for editing, creates a rectangle in the chart, makes a copy of the rectangle, then deactivates editing.

```
{GraphEdit Chart1, 1}  
{CreateObject Rect, 147, 176, 416, 427}  
{Duplicate 269, 338}  
{GraphDeactivate}
```

### ► Related topics

## {GraphDelete}

### Syntax

{GraphDelete *Name*}

### PerfectScript Syntax

GraphDelete (Name:String)

### Description

{GraphDelete} deletes the specified chart from the active notebook.

You can use {GraphDelete?} or {GraphDelete!} to display the Delete Chart dialog box. {GraphDelete?} lets you manipulate the dialog box, whereas {GraphDelete!} relies on the macro to manipulate it.

### Parameters

<i>Name</i>	Name of the chart to delete
-------------	-----------------------------

## {GraphEdit}

### Syntax

{GraphEdit *Name*,<*InPlace?*{0|1}}

### PerfectScript Syntax

GraphEdit (Name:String; [InPlace?:Enumeration {Yes!; No!}])

### Description

{GraphEdit} displays the specified chart in a chart window for editing. Use the *InPlace?* argument to edit a floating chart on the notebook sheet.

You can use {GraphEdit?} or {GraphEdit!} to display the Edit Chart dialog box. {GraphEdit?} lets you manipulate the dialog box, whereas {GraphEdit!} relies on the macro to manipulate it.

### Example

The following macro selects the floating chart named Inserted1 and then activates its source chart (Chart1) for editing on the notebook sheet.

```
{SelectFloat Inserted1}
{GraphEdit Chart1,1}
```

### Parameters

<i>Name</i>	Name of the chart to edit
<i>InPlace?</i>	Whether to edit the chart in place on a notebook sheet; 0 = no, 1 = yes; the default is 0

## {GraphGallery}

### Syntax

{GraphGallery *ChartStyle*, *ColorScheme*}

### PerfectScript Syntax

GraphGallery (GraphStyle:String; ColorScheme:String)

### Description

{GraphGallery} applies a chart style and color scheme to selected charts.

Available choices for *ColorScheme* are:

No change      Pastels

Default Fire and Ice

Grayscale      Bright and Bold

Icy Blues	Color Washes
Deep Reds	Black and White Patterns
Autumn Leaves	Color Patterns
Tangerine	Tiled Men

You can use {GraphGallery?} or {GraphGallery!} to display the Chart Gallery dialog box. {GraphGallery?} lets you manipulate the dialog box, whereas {GraphGallery!} relies on the macro to manipulate it.

### Example

The following macro selects a 3-D Bar chart style and a "Tangerine" color scheme:

```
{GraphGallery "3dbar", "Tangerine"}
```

### Parameters

<i>ChartStyle</i>	The style of chart; see <a href="#">{GraphSettings.Type}</a> for a list of chart types
<i>ColorScheme</i>	The color scheme used for the chart

## {GraphNew}

### Syntax

```
{GraphNew Name,<UseCurrentBlock?(0|1)>}
```

### PerfectScript Syntax

```
GraphNew (Name:String; [UseCurrentBlock?:Enumeration {Yes!; No!}])
```

### Description

{GraphNew} creates a new chart and displays it in a chart window. If *UseCurrentBlock?* is 1, any selected data is shown in the chart; if it is 0, {GraphNew} creates a new chart without data.

You can use {GraphNew?} or {GraphNew!} to display the New Chart dialog box. {GraphNew?} lets you manipulate the dialog box, whereas {GraphNew!} relies on the macro to manipulate it.

### Parameters

<i>Name</i>	Name of the new chart
<i>UseCurrentBlock?</i>	Whether to chart the current selected cells; 0 = no, 1 = yes; the default is 0

## {GraphSettings.Check}

### Syntax

```
{GraphSettings.Check}
```

### PerfectScript Syntax

```
GraphSettings_Check ()
```

## **{GraphSettings.Reset}**

### **Syntax**

{GraphSettings.Reset}

### **PerfectScript Syntax**

GraphSettings\_Reset ()

## **{GraphSettings.Titles}**

### **Syntax**

{GraphSettings.Titles *Main, Sub, X-Axis, Y-Axis, Y2-Axis*}

### **PerfectScript Syntax**

GraphSettings\_Titles (Main:String; Sub:String; XAxis:String; YAxis:String; Y2Axis:String)

### **Description**

{GraphSettings.Titles} sets the titles of the active chart (or selected floating chart or chart icon). Each argument is a string; to reset a title, use an empty string ("").

### **Example**

The following macro command displays the chart Profit99 in a chart window and sets its main title and subtitles. The empty strings (") indicate that there are no axis titles.

```
{GraphEdit Profit99}
```

```
{GraphSettings.Titles "Projected Profits", "1999", "", "", ""}
```

### **Parameters**

<i>Main</i>	Main title of the chart
<i>Sub</i>	Title appearing below the main title of the chart
<i>X-Axis</i>	Title of the chart's x axis
<i>Y-Axis</i>	Title of the chart's y axis
<i>Y2-Axis</i>	Title of the chart's secondary y axis

### **► Related topics**

## {GraphSettings.Type}

### Syntax

{GraphSettings.Type *Type,Class*}

### PerfectScript Syntax

GraphSettings\_Type (Type:String)

### Description

{GraphSettings.Type} lets you specify how the data in a chart is displayed. It affects the active chart (or chart icon or floating chart). *Class* specifies the class of chart type being used. *Class* can be one of six settings: Area/Line, Bar, Stacked Bar, Pie, Specialty, and Text.

These are the chart types you can choose:

```
{GraphSettings.Type "3D Area,Area/Line"}
{GraphSettings.Type "3D Marker,Area/Line"}
{GraphSettings.Type "3D Ribbon,Area/Line"}
{GraphSettings.Type "3D Unstacked area,Area/Line"}
{GraphSettings.Type "Area,Area/Line"}
{GraphSettings.Type "Line,Area/Line"}
{GraphSettings.Type "Rotated area,Area/Line"}
{GraphSettings.Type "Rotated line,Area/Line"}
{GraphSettings.Type "2DHalf bar,Bar"}
{GraphSettings.Type "3D Bar,Bar"}
{GraphSettings.Type "3D Step,Bar"}
{GraphSettings.Type "Area_bar,Bar"}
{GraphSettings.Type "Bar,Bar"}
{GraphSettings.Type "Hilo_bar,Bar"}
{GraphSettings.Type "Line_bar,Bar"}
{GraphSettings.Type "Multiple bar,Bar"}
{GraphSettings.Type "R2D bar,Bar"}
{GraphSettings.Type "R2DHalf bar,Bar"}
{GraphSettings.Type "R3D bar,Bar"}
{GraphSettings.Type "Variance,Bar"}
{GraphSettings.Type "3D Column,Pie"}
{GraphSettings.Type "3D Doughnut,Pie"}
{GraphSettings.Type "3D Pie,Pie"}
{GraphSettings.Type "Column,Pie"}
{GraphSettings.Type "Doughnut,Pie"}
{GraphSettings.Type "Multiple 3D columns,Pie"}
{GraphSettings.Type "Multiple 3D pies,Pie"}
{GraphSettings.Type "Multiple columns,Pie"}
{GraphSettings.Type "Multiple pies,Pie"}
{GraphSettings.Type "Pie,Pie"}
{GraphSettings.Type "3D Contour,Specialty"}
{GraphSettings.Type "3D ShadedSurface,Specialty"}
{GraphSettings.Type "3D Surface,Specialty"}
{GraphSettings.Type "HiLo,Specialty"}
{GraphSettings.Type "Polar radar,Specialty"}
{GraphSettings.Type "XY,Specialty"}
{GraphSettings.Type "100 stacked bar,Stacked Bar"}
{GraphSettings.Type "100 stacked line,Stacked Bar"}
{GraphSettings.Type "3D100 stacked bar,Stacked Bar"}
{GraphSettings.Type "3D Stacked bar,Stacked Bar"}
{GraphSettings.Type "R2D100 stacked bar,Stacked Bar"}
{GraphSettings.Type "R2D100 stacked line,Stacked Bar"}
{GraphSettings.Type "R2D stacked bar,Stacked Bar"}
{GraphSettings.Type "R2D stacked line,Stacked Bar"}
{GraphSettings.Type "R3D100 stacked bar,Stacked Bar"}
{GraphSettings.Type "R3D stacked bar,Stacked Bar"}
{GraphSettings.Type "Stacked bar,Stacked Bar"}
{GraphSettings.Type "Stacked line,Stacked Bar"}
{GraphSettings.Type "Blank,Text"}
```

```
{GraphSettings.Type "Bullet,Text"}
```

### **Example**

{GraphSettings.Type "3-D Pie,Pie"} make the active chart a 3-D pie chart.

► **Related topics**



## {**GraphView**}

### Syntax

```
{GraphView <ChartName1,ChartName2,...>}
```

### PerfectScript Syntax

```
GraphView (GraphName:String; {[MoreGraphName:String]})
```

### Description

{GraphView} displays a full-screen chart (or series of charts). {GraphView} without an argument displays the active chart (or chart icon or floating chart).

You can use {GraphView?} or {GraphView!} to display the View Chart dialog box. {GraphView?} lets you manipulate the dialog box, whereas {GraphView!} relies on the macro to manipulate it.

### Example

The following macro displays the named charts Profit90 through Profit94.

```
{GraphView Profit90,Profit91,Profit92,Profit93,Profit94}
```

### Parameters

<i>ChartName</i>	Name of the first chart to display
<i>1</i>	(optional)
<i>ChartName</i>	Name of the second chart to display
<i>2</i>	(optional)

## {**GraphWindow**}

### Syntax

```
{GraphWindow.Property}
```

### PerfectScript Syntax

```
GraphWindow_Aspect_Ratio (Mode:String)
```

```
GraphWindow_Grid (Settings:String)
```

### Description

{GraphWindow} is equivalent to right-clicking the title bar of a chart window to set its Aspect Ratio or Grid properties.

{GraphWindow.Aspect\_Ratio *Option*} sets the aspect ratio of the active chart. *Option* can be one of the following settings: "35mm Slide," "Floating Chart," "Full Extent," "Printer Preview," or "Screen Slide."

{GraphWindow.Grid *GridSize,DisplayGrid,SnapToGrid*} sets the grid size of the active chart window. Use *GridSize* to specify the percent of chart window between grid points; *DisplayGrid* specifies whether the grid is visible; *SnapToGrid* specifies whether the grid is active.

### Example

This macro sets up a 10 by 10 grid, displays the grid, and enables it.

```
{GraphWindow.Grid "10, Yes, Yes"}
```

## {Group}

### Syntax

{Group.Option}

### PerfectScript Syntax

Group\_Define (GroupName:String; StartPage:String; EndPage:String)

Group\_Delete (GroupName:String)

Group\_ResetNames ()

### Description

{Group} creates and deletes sheet groups.

Once you have defined a sheet group, you can use {Notebook.Group\_Mode "On"} to activate Group mode. Use "Off" to cancel Group mode.

You can use {Group?} or {Group!} to display the Define/Modify Group dialog box. {Group?} lets you manipulate the dialog box, whereas {Group!} relies on the macro to manipulate it.

### Options

{Group.Define <i>GroupName, StartPage, EndPage</i> }	Creates sheet groups
{Group.Delete <i>GroupName</i> }	Deletes the currently selected sheet group
{Group.ResetNames}	Clears all group names in the notebook

## {GroupObjects}

### Description

{GroupObjects} groups selected objects in a chart window so they can be treated as one object in subsequent operations. Use {UngroupObjects} to treat them independently again.

### ▶ Related topics



## **{HELP}**

### **Description**

{HELP} is equivalent to the Help key, F1. It displays a help topic.

### **▶ Related topics**

## **{HideErrorMessage}**

### **Syntax**

{HideErrorMessage}

### **PerfectScript Syntax**

HideErrorMessage ()

### **Description**

Suppresses the ability for Quattro Pro to show an error message, if one is warranted.

### **▶ Related topics**

## {HISTOGRAM}

### Syntax

{HISTOGRAM *InBlock*, *OutBlock*, <*BinBlock*>, <*Pareto*>, <*Cum*>}

### PerfectScript Syntax

HISTOGRAM (InBlock:String; OutBlock:String; [BinBlock:String]; [Pareto?:Enumeration {Yes!; No!}]; [Cum?:Enumeration {Yes!; No!}])

### Description

{HISTOGRAM} calculates the probability and cumulative distributions for a sample population, based on a series of bins. {HISTOGRAM} is equivalent to the Histogram analysis tool.

### Parameters

<i>InBlock</i>	Input cells containing one or more columns or rows of numeric values; the cells must not contain labels
<i>OutBlock</i>	Upper-left cell of the output cells
<i>BinBlock</i>	Set of numbers defining the bin ranges; BinBlock numbers must be in ascending order; if BinBlock is omitted, bins are distributed evenly from the minimum to the maximum values in InBlock, with the number of bins equal to the square root of the number of values in InBlock
<i>Pareto</i>	1 to arrange the output table in both descending frequency order and ascending BinBlock order; 0 to arrange the output table in ascending BinBlock order; the default is 0
<i>Cum</i>	Flag indicating whether to generate a column in OutBlock showing cumulative percentages: yes (1) or no (0); the default is 0

### ► Related topics

## **{HLINE}**

### **Syntax**

{HLINE *Distance*}

### **PerfectScript Syntax**

HLine (Distance:Numeric)

### **Description**

{HLINE} scrolls the active notebook horizontally by *Distance* columns. If the number is positive, it scrolls right; if negative, it scrolls left. {HLINE} does not 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.

### **Parameter**

{HLINE 10} scrolls the display 10 columns to the right.

{HLINE -5} scrolls the display 5 columns to the left.

### **Parameters**

Distance      Distance in columns to scroll the active notebook horizontally

### **► [Related topics](#)**

## **{Home}**

### **Description**

{HOME} is equivalent to the Home key.

▶ **Related topics**



## **{HPAGE}**

### **Syntax**

{HPAGE *Distance*}

### **PerfectScript Syntax**

HPage (Distance:Numeric)

### **Description**

{HPAGE} scrolls the active notebook horizontally by *Distance* screens. If the number is positive, it scrolls right; if negative, it scrolls left. {HPAGE} does not move the selector; only the view of the notebook is altered. Use [{BIGRIGHT}](#) or [{BIGLEFT}](#) to move the selector horizontally.

### **Parameters**

*Distance*      Distance in screens to scroll the active notebook horizontally

### **▶ [Related topics](#)**



## {IF}

### Syntax

{IF *Condition*}

### Description

{IF} operates like @!F. {IF} evaluates *Condition*; if it is numeric and nonzero, it is considered to be TRUE and macro execution continues in the same cell; if *Condition* is anything else, it is considered FALSE, and macro execution continues in the cell directly below the {IF} command. Unlike @!F, {IF} commands cannot be nested within each other.

### Example

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 do not want both the true and false clauses executing, be sure to include `{BRANCH}` or `{QUIT}` in the same cell as {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}
```

### Parameters

<i>Condition</i>	A logical expression (or the address of a cell containing a label, value, or expression)
------------------	--

### ► Related topics

## {IFAUTOOBJ}

### Syntax

{IFAUTOOBJ *ObjectExpression*}

### Description

{IFAUTOOBJ} is like {IF}, but instead evaluating a logical condition it expects the object expression resulting from an OLE automation object. If *ObjectExpression* is true, {IFAUTOOBJ} runs the macro in the same cell; otherwise, it skips to the next cell.

### Example

```
{ASSIGN OLE2App, GetObject("DispCalc.Application")}  
{IFAUTOOBJ OLE2App}{BRANCH A10}  
{ASSIGN OLE2App, CreateObject("DispCalc.Application")}
```

### Parameters

<i>ObjectExpression</i>	Object expression resulting from an OLE automation object
-------------------------	---

## {IFKEY}

### Syntax

{IFKEY *String*}

### Description

{IFKEY} 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}). Do not 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 is not the name of a macro command. *String* can be a string or a text formula.

### Parameters

<i>String</i>	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
---------------	---

### ► Related topics

## {IMFORMAT}

### Syntax

{IMFORMAT *Format*}

### PerfectScript Syntax

IMFORMAT (Format:Numeric)

### Description

{IMFORMAT} specifies how complex numbers display in the active notebook, and returns a label showing the selected format.

### Example

{IMFORMAT 1} returns "x+iy"

{IMFORMAT 2} returns "x+jy"

### Parameters

<i>Forma</i> <i>t</i>	Flag indicating what suffix and format to use for imaginary coefficient of complex number; the default is 1; 1 = $x + yi$ , 2 = $x + yj$ , 3 = $x + iy$ , 4 = $x + jy$
--------------------------	--

## {ImportGraphic}

### Syntax

{ImportGraphic *Filename*}

### PerfectScript Syntax

ImportGraphic (Filename:String)

### Description

{ImportGraphic} imports graphics files into a chart window.

You can use {ImportGraphic?} or {ImportGraphic!} to display the Insert Image dialog box. {ImportGraphic?} lets you manipulate the dialog box, whereas {ImportImage!} relies on the macro to manipulate it.

### Parameters

<i>Filenam</i> <i>e</i>	Name of the bitmap or other graphics file to import
----------------------------	---

## {ImportGraphic.Clipart}

### Syntax

{ImportGraphic.Clipart}

### PerfectScript Syntax

ImportGraphic Clipart()

### Description

Equivalent to Insert ► Graphics  
► Clipart

## **{INDICATE}**

### **Syntax**

{INDICATE *String*}

### **PerfectScript Syntax**

Indicate ([String:String])

### **Description**

{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 ""}.

### **Example**

{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.

### **Parameters**

*String*      Any seven-character string

### **▶ Related topics**

## **{InitApp}**

### **Syntax**

{InitApp}

### **PerfectScript Syntax**

InitApp ()

### **Description**

Starts Quattro Pro.

## **{INITIATE}**

### **Syntax**

{INITIATE *AppName,Topic,ChannelLoc*}

### **Description**

{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 (DDE communication) is stored in the cell *ChannelLoc*. *Topic* can be a Corel WordPerfect document, an Excel spreadsheet, an ObjectVision form, 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.

You can use {INITIATE} multiple times with the same application; if you do this, use a different *ChannelLoc* for each conversation. Once a channel opens, you can use [{REQUEST}](#), [{EXECUTE}](#), or [{POKE}](#) to communicate with the application.

When the DDE conversation is complete, use [{TERMINATE}](#) to end it. See {EXECUTE} for an example of {INITIATE}.

### **Parameters**

<i>AppName</i>	Name of the Windows application to communicate with
<i>Topic</i>	Name of the file within the Windows application to communicate with
<i>ChannelLoc</i>	Cell to contain the Channel ID number of the conversation if communication succeeds

### **▶ Related topics**



## **{INS}, {INSERT}, {INSOFF}, and {INSON}**

### **Description**

{INS} and {INSERT} toggle the Ins key on or off. {INSOFF} is equivalent to Ins off, and {INSON} to Ins on.

▶ **Related topics**

## **{InsertBreak}**

### **Syntax**

{InsertBreak}

### **PerfectScript Syntax**

InsertBreak ()

### **Description**

Inserts a new line and a hard page break into notebook print blocks at the current selector location.

## **{InsertObject}**

### **Syntax 1: Embedding/Linking from a File**

{InsertObject *Filename*, <*DisplayAsIcon*?(0|1)>, <*Linked*?(0|1)>}

### **Syntax 2: Embedding a New Object**

{InsertObject *ObjectType*, <*DisplayAsIcon*?(0|1)>}

### **PerfectScript Syntax**

InsertObject (ObjectTypeOrFilename:String; [DisplayAsIcon?:Enumeration {Yes!; No!}]; [Linked?:Enumeration {Yes!; No!}])

### **Description**

{InsertObject} inserts an OLE object into the active notebook without using the Clipboard.

You can use {InsertObject?} or {InsertObject!} to display the Insert Object dialog box. {InsertObject?} lets you manipulate the dialog box, whereas {InsertObject!} relies on the macro to manipulate it.

### **Example**

This macro inserts a picture created in Paintbrush into the active notebook.

```
{InsertObject "Paintbrush Picture"}
```

### **Parameters 1**

<i>Filename</i>	File that you want to link/embed as an object
<i>DisplayAsIcon</i> <i>n</i>	Whether to display the object as an icon; 0 to show the object as it looks in the server application; 1 to display the object as an icon
<i>Linked?</i>	Whether to link to the file; 0 to not link; 1 to link; the default is 0

### **Parameters 2**

<i>ObjectType</i>	Type of object to insert (the name of an OLE server)
<i>DisplayAsIcon</i> <i>n</i>	Whether to display the object as an icon; 0 to show the object as it looks in the server application; 1 to display the object as an icon

## **{InsertObject.DrawPicture}**

### **Syntax**

{InsertObject.DrawPicture}

### **PerfectScript Syntax**

InsertObject\_DrawPicture ()

## **{InsertObject.TextArt}**

### **Syntax**

{InsertObject.TextArt}

### **PerfectScript Syntax**

InsertObject\_TextArt ()

### **Description**

Allows you to insert TextArt.

## **{InsertPageBreak}**

### **Syntax**

{InsertPageBreak.*Option*}

### **PerfectScript Syntax**

InsertPageBreak\_Create (Row:Numeric; Column:Numeric)

InsertPageBreak\_Delete (Row:Numeric; Column:Numeric)

### **Description**

{InsertPageBreak.*Create*} inserts a page break above Row# and to left of Column#.

{InsertPageBreak.*Delete*} deletes the current PageBreak above Row# and to left of Column#.

### **Options**

{InsertPageBreak.Create Row#, Column#}	Creates a hard page break to start a new page
{InsertPageBreak.Delete Row#, Column#}	Deletes a hard page break

## **{INSPECT}**

### **Description**

{INSPECT} is equivalent to the Inspect key, F12. It displays an Object Inspector for the current object.

### **► Related topics**

# {Invert}

## Syntax

{Invert.Option}

## PerfectScript Syntax

Invert\_Destination (Block:String)

Invert\_Go ()

Invert\_Source (Block:String)

## Description

{Invert} inverts a square matrix (indicated by {Invert.Source *Block*}) and stores the invert matrix in other cells (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.

You can use {Invert?} or {Invert!} to display the Matrix Invert dialog box. {Invert?} lets you manipulate the dialog box, whereas {Invert!} relies on the macro to manipulate it.

## Options

{Invert.Destination <i>Block</i> }	Specifies the upper-left cell of the area where you want to write the inverted matrix
{Invert.Go}	Inverts the selected matrix
{Invert.Source <i>Block</i> }	Specifies the matrix you want to invert

## ► Related topics

## **{IsAutoObj}**

### **Syntax**

{IsAutoObj *ObjectI*}

### **PerfectScript Syntax**

IsAutoObj (Object: String)





## **{LEFT} and {L}**

### **Syntax**

{LEFT <Number>} or {L <Number>}

### **Description**

{LEFT} and {L} are equivalent to the Left-arrow key. The optional argument *Number* moves the selector the corresponding number of columns to the left. You can use cell references or cell names as arguments.

### **Example**

{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 cells named *count*.

### **Parameters**

*Number* Any positive integer (optional)

### **▶ Related topics**

# {LET}

## Syntax

{LET *Location*,*Value*<:*Type*>}

## PerfectScript Syntax

Let (Cell:String; Value:Any)

## Description

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 *Type*, 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.

*Value* cannot be an @ARRAY formula. {LET} does not enter array values. Use {PUTCELL} or {PUTCELL2} to enter array values.

You can use {LET} to invoke add-in @functions or macros contained in DLLs. Specify the add-in as *Value*, using this syntax for functions:

```
@dllname.functionname(functionargument1, functionargument2, ...)
```

For example, this statement calls the @function MEDIAN, included in 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)}
```

The macro syntax is identical:

```
@dllname.macroname(macroargument1, macroargument2, ...)
```

## Example

{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)

## Parameters

<i>Location</i>	Cell in which to store the specified value
<i>Value</i>	Numeric or string value to be stored in <i>Location</i>
<i>Type</i>	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)

## ► Related topics

## **{Link}**

### **Syntax**

```
{Link "string"}
```

### **Description**

{Link} applies a link to the currently selected object, such as a pushbutton in a dialog or toolbar.

### **Example**

```
{Link "ON Clicked DOMACRO {filesave}"}
```

## {Links}

### Syntax

{Links.*Option*}

### PerfectScript Syntax

Links\_Change (OldName:String; NewName:String)

Links\_Delete (LinkName:String)

Links\_Open (LinkName:String)

Links\_Refresh (LinkName:String)

### Description

{Links.*Option*} refreshes, changes, or deletes links in the active notebook.

*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).

### Example

{Links.Refresh \*} refreshes all links in the active notebook.

The following macro displays the Open Links dialog box and lets you select the name of a linked notebook to open.

```
{Links.Open}  
{ PAUSEMACRO }
```

### Options

{Links.Change <i>OldName, NewName</i> }	Switches links from one file to another
{Links.Delete <i>LinkName</i>   *} (* = all links)	Deletes notebook links
{Links.Open <i>LinkName</i>   *} (* = all links)	Opens files linked to the active notebook
{Links.Refresh <i>LinkName</i>  *} (* = all links)	Refreshes links to unopened files

## {LOOK}

### Syntax

{LOOK *Location*}

### Description

When Quattro Pro runs a macro, it does not respond to keystrokes you enter (except Ctrl+Break). If you press 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 next example).

{LOOK} does not 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}.

### Example

In the following example, the macro gives you 15 seconds to choose the next menu choice. If you do not, you must reenter the password. **Important:** Type the line to the right of `_check_it` with no hard returns until after {BRANCH \_password}, then press Enter to insert it into one cell (the macro commands from {LOOK keystroke} to {BRANCH \_password} are shown here on separate lines for readability).

```
\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

```

### **Parameters**

*Location*      A cell in which to store a typed character

### **► Related topics**



## **{MACROS}**

### **Description**

{MACROS} is equivalent to the Macros key, Shift+F3, which displays a menu of macro commands to type into the input line.

▶ **Related topics**

## **{MapExpert}**

### **Description**

{MapExpert} displays the first Map Expert dialog box. The macro has no arguments.



## **{MapNew}**

### **Syntax**

{MapNew *Name*, <*UseBlock*>}

### **Description**

{MapNew} creates maps that display data, using colors or patterns to distinguish regions from each other.

### **Parameters**

<i>Name</i>	Name of the new map
<i>UseBloc</i>	1 to create the map using data in the current cells
<i>k</i>	

### **▶ Related topics**

## **{MARK}**

### **Description**

{MARK} is equivalent to the Select key, Shift+F7, which lets you begin selection of cells.

▶ **Related topics**

## **{MCORREL}**

### **Syntax**

{MCORREL *InBlock*, *OutBlock*, <*Grouped*>, <*Labels*>}

### **PerfectScript Syntax**

MCORREL (InBlock:String; OutBlock:String; [Grouped:String]; [Labels?:Enumeration {Yes!; No!}])

### **Description**

{MCORREL} computes the correlation matrix between two or more data sets. {MCORREL} is equivalent to the Correlation analysis tool.

### **Parameters**

<i>InBlock</i>	Input cells containing two or more sets of numeric data arranged in columns or rows
<i>OutBlock</i>	Upper-left cell of the output cells
<i>Grouped</i>	"C" to group results by column or "R" to group results by row; the default is "C"
<i>Labels</i>	1 if labels are located in the first column or row of the input cells; 0 if the input cells do not contain labels; the default is 0

### **► Related topics**

## **{MCOVAR}**

### **Syntax**

{MCOVAR *InBlock*, *OutBlock*, <*Grouped*>, <*Labels*>}

### **PerfectScript Syntax**

MCOVAR (InBlock:String; OutBlock:String; [Grouped:String]; [Labels?:Enumeration {Yes!; No!}])

### **Description**

{MCOVAR} returns the covariance matrix between two or more data sets. {MCOVAR} is equivalent to the Covariance analysis tool.

### **Parameters**

<i>InBlock</i>	Input cells containing two or more sets of numeric data arranged in columns or rows
<i>OutBlock</i> <i>k</i>	upper-left cell of the output cells
<i>Grouped</i>	"C" to group results by column or "R" to group results by row; the default is "C"
<i>Labels</i>	1 if labels are located in the first column or row of the input cells; 0 if the input cells do not contain labels; the default is 0

### **► Related topics**

## **{MENU}**

### **Description**

{MENU} is equivalent to the Menu key (Alt or F10). This command moves the selector to the menu bar and highlights the first menu.

► **Related topics**

## {MENUBRANCH}

### Syntax

{MENUBRANCH *Location*}

### Description

{MENUBRANCH} pauses macro execution to display the custom pop-up menu stored at *Location*. After you choose 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 you press Esc or click outside the menu; in this case, the macro continues in the {MENUBRANCH} cell.

With the exception of the Esc key, a custom pop-up menu acts the same as one of Quattro Pro's own menus. You can press the arrow keys to look at each item, and can choose a menu item by pressing Enter while highlighting it, pressing the first letter of the menu item's name, or choosing it with the mouse.

### Example

The following macro displays a custom menu that offers you three choices. If you press Esc, the menu is redisplayed with {BRANCH \F}.

```
quit_menu
ContinueReturn
Quit
Continue macro
Return to Ready
Leave Quattro Pro
{BRANCH \F}{BRANCH _continue}      {MENUBRANCH sure}
sure No      Yes
Stay in Quattro Pro      Return to DOS
{BRANCH \F}{FileExit}
\F      {MENUBRANCH quit_menu}
      {BRANCH \F}_continue
```

### Parameters

*Location*      cells containing menu cells

#### ► Note

- This command is obsolete

#### ► Related topics

## {MENUCALL}

### Syntax

{MENUCALL *Location*}

### Description

{MENUCALL} pauses macro execution and displays the custom pop-up menu stored at *Location*. It treats the called menu as a subroutine. After you choose an item from the menu, macro execution continues in the cell containing {MENUCALL}.

See [{ADDMENU}](#) and [{ADDMENUITEM}](#) for information on adding menus or menu items to the Quattro Pro menu bar.

### Example

The following macro uses two consecutive custom menus to let you change search criteria for a database. It then copies records that meet the criteria to the output cells and branches to another macro to print the output cells as labels. The cell 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 Cells:

```
STATUS   PAID
A       F
```

### Parameters

*Location*      Cells containing a custom Quattro pro menu

### ► [Related topics](#)

## {MESSAGE}

### Syntax

{MESSAGE *Block,Left,Top,Time*}

### Description

{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 chart, bitmap, or macro button, its image displays in the message cells.

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 to 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 is not a notebook (for example, a chart window).

If you enter 0 for *Time*, Quattro Pro displays the box until you press any key. (You can use {GRAPHCHAR} to test for which key is pressed.)

### Example

This example displays a message box for 15 seconds and then displays another box indefinitely. If you press Y, the macro closes the notebook without saving it. If you press N (which is returned in the cells *the\_key*), you remove the second message box.

*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}

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...

_lose_data  {FileClose 0}
```

### Parameters

<i>Block</i>	Cell name or coordinates of the text and/or floating object to display in the message box
<i>Left</i>	Screen column number (counting from 0) where the top-left corner of the box should appear
<i>Top</i>	Screen line number (counting from 0) where the top left corner of the box should appear
<i>Time</i>	Quattro Pro time serial number

### ► Related topics



## **{MOVEAVG}**

### **Syntax**

{MOVEAVG *InBlock*, *OutBlock*, <*Interval*>, <*StdErrs*>}

### **PerfectScript Syntax**

MOVEAVG (*InBlock*:String; *OutBlock*:String; *Interval*:Numeric; [*StdErrs*?:Enumeration {Yes!; No!}])

### **Description**

{MOVEAVG} returns a moving average for a specified *Interval* based on the values for the preceding periods in *InBlock*. {MOVEAVG} is equivalent to the Moving Average analysis tool.

### **Parameters**

<i>InBlock</i>	Input cells containing a single column or row with at least four numeric values; the cells must not contain labels
<i>OutBlock</i>	Upper-left cell of the output cells
<i>Interval</i>	Number of values to include in the moving average; the default is 3
<i>StdErrs</i>	Flag indicating whether to include standard error values in the <i>OutBlock</i> : yes (1) or no (0); the default is 0

### **► Related topics**

# {MOVETO}

## Syntax

{MOVETO x,y}

## PerfectScript Syntax

MoveTo (x:Numeric; y:Numeric)

## Description

{MOVETO} moves all selected objects in the active window (dialog, chart, or Objects sheet window) 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 chart window. It also moves chart icons on the Objects sheet. (Use {FLOATMOVE} to move floating objects in a notebook window.)

The coordinates *x* and *y* represent where to move the upper-left corner of the object(s). Object size does not change.

## Parameters

*x,y*

Position to move the currently selected object(s) to in pixels

## ▶ Related topics

# {MTGAMT}

## Syntax

{MTGAMT <OutBlock>, <Rate>, <Term>, <OrigBal>, <EndBal>, <LastYear>}

## PerfectScript Syntax

MTGAMT ([OutBlock:String]; [Rate:Numeric]; [Term:Numeric]; [OrigBal:Numeric]; [EndBal:Numeric]; [LastYear:Numeric])

## Description

{MTGAMT} generates an amortization schedule for a mortgage. {MTGAMT} is equivalent to the Amortization Schedule analysis tool.

## Parameters

<i>OutBlock</i>	Upper-left cell of the output cells
<i>Rate</i>	Yearly interest rate; the default is 0.12
<i>Term</i>	Number of years in the loan; the default is 30 years; can be a fractional value to designate months (for example, 3+5/12)
<i>OrigBal</i>	Original loan balance; the default is \$100,000
<i>EndBal</i>	Balance at loan completion; the default is \$0
<i>LastYear</i>	Last year through which the amortization period is generated; the default is equal to Term (the end of the loan); can be a fractional value to designate months (for example, 3+5/12)

## ► Related topics

## **{MTGREFI}**

### **Syntax**

{MTGREFI <OutBlock>, <CurrBal>, <CurrRate>, <RemTerm>, <CandPctFees>, <CandRate>}

### **PerfectScript Syntax**

MTGREFI (OutBlock:String; [CurrBal:Numeric]; [CurrRate:Numeric]; [RemTerm:Numeric]; [CandPctFees:Numeric]; [CandRate:Numeric])

### **Description**

{MTGREFI} generates a table of information relating to refinancing a mortgage. {MTGREFI} is equivalent to the Mortgage Refinancing analysis tool.

### **Parameters**

<i>OutBlock</i>	Upper-left cell of the output cells
<i>CurrBal</i>	Remaining principal on the current loan
<i>CurrRate</i>	Annual interest rate on the current loan
<i>RemTerm</i>	Remaining term on the current loan
<i>CandPctFee</i>	Percentage fees ("points") for the candidate loan
<i>s</i>	
<i>CandRate</i>	Annual interest rate for the candidate loan

### **► Related topics**

# {Multiply}

## Syntax

{Multiply.Option}

## PerfectScript Syntax

Multiply\_Destination (Block:String)

Multiply\_Go ()

Multiply\_Matrix\_1 (Block:String)

Multiply\_Matrix\_2 (Block:String)

## Description

{Multiply} multiplies one matrix ({Multiply.Matrix\_1 Block}) by another ({Multiply.Matrix\_2 Block}) and stores the product in other cells ({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.

You can use {Multiply?} or {Multiply!} to display the Matrix Multiply dialog box. {Multiply?} lets you manipulate the dialog box, whereas {Multiply!} relies on the macro to manipulate it.

## Example

This macro multiplies cells C2..D6 by cells C18..G19 and stores the results in the cells with upper-left cell F1.

```
{Multiply.Matrix_1 A:C2..D6}
```

```
{Multiply.Matrix_2 A:C18..G19}
```

```
{Multiply.Destination A:F1}
```

```
{Multiply.Go}
```

## Options

{Multiply.Destination Block}	Specifies the top-left cell of the area where you want to write the resulting matrix
{Multiply.Go}	Executes the multiplication
{Multiply.Matrix_1 Block}	Specifies the first matrix to multiply
{Multiply.Matrix_2 Block}	Specifies the second matrix to multiply

## ► Related topics



## **{NAME}**

### **Description**

{NAME} is equivalent to the Choices key, F3, which displays a list of cell names in the current notebook, if cell names exist in the notebook. (If there are no named cells, the list of cell names won't appear.)

Use {NAME} with {GOTO}.

### **Example**

{GOTO} {NAME}

### **▶ Related topics**

## {NamedStyle}

### Syntax

{NamedStyle.Option}

### PerfectScript Syntax

NamedStyle\_Alignment (Settings:String)

NamedStyle\_Define (StyleName:String; Align?:Enumeration {Yes!; No!}; NumericFormat?:Enumeration {Yes!; No!}; Protection?:Enumeration {Yes!; No!}; Lines?:Enumeration {Yes!; No!}; Shading?:Enumeration {Yes!; No!}; Font?:Enumeration {Yes!; No!}; TextColor?:Enumeration {Yes!; No!})

NamedStyle\_Delete (StyleName:String)

NamedStyle\_Font (Settings:String)

NamedStyle\_Line\_Drawing (Settings:String)

NamedStyle\_Numeric\_Format (Settings:String)

NamedStyle\_Protection (Settings:String)

NamedStyle\_Shading (Settings:String)

NamedStyle\_Text\_Color (ColorID:Numeric)

### Description

{NamedStyle} lets you create styles in the active notebook.

These command equivalents do not 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. *Bold*, *Italic*, *Underline* and *Strikeout* can be "Yes" to include that type feature or "No" to omit it.

{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").

You can use {NamedStyle?} or {NamedStyle!} to display the Styles dialog box. {NamedStyle?} lets you manipulate the dialog box, whereas {NamedStyle!} relies on the macro to manipulate it.

### Example

This macro creates a new style named RedNote, which makes the active cells red, and sets a new font.

```
{NamedStyle.Font "Courier,10,Yes,No,No,No"}
```

```
{NamedStyle.Text_Color "4"}
```

```
{NamedStyle.Define RedNote,0,0,0,0,0,0,1,1}
```

### ▶ Related topics



## {Navigate}

### Syntax

{Navigate.Option}

### PerfectScript Syntax

Navigate\_GoTo (Where:Enumeration {Up!; Left!; Right!; Down!; TopLeft!; BottomLeft!; TopRight!; BottomRight!}; [Extend?:Enumeration {Yes!; No!}])


Navigate\_Jump (Where:Enumeration {Up!; Left!; Right!; Down!})

Navigate\_SelectTable ()

Navigate\_Zoom2Fit ()

### Description

{Navigate} is equivalent to the navigation tools available on the Data Manipulation Toolbar.

{Navigate.SelectTable} is equivalent to the SpeedSelect button  on the Data Manipulation Toolbar, which expands selection from a cell or cells within a table to the entire table. {Navigate.Zoom2Fit} is equivalent to the Zoom To Fit button



. {Navigate.GoTo} performs the same actions as the Top Left Of Table



, Top Right Of Table



, Bottom Left Of Table



, and Bottom Right Of Table



buttons. {Navigate.Jump} jumps to the next table or to the selected boundary of the current table.

### Example

The following macro selects cell C6 in the table below, then selects the entire table that C6 belongs to, and zooms to fit the table on the page.

```
{SelectBlock A:C6}
```

```
{Navigate.SelectTable}
```

```
{Navigate.Zoom2Fit}
```

	A	B	C	D
1		Sales	Expenses	Profits
2	Jan	1580	700	880
3	Feb	2474	545	1929
4	Mar	2570	656	1914
5	Apr	2876	454	2422
6	May	3223	489	2734
7	Jun	2987	470	2517
8	Jul	3178	500	2678

### Options

{Navigate.SelectTable}

Expands selection to the table boundaries

{Navigate.Zoom2Fit}

Zooms so that a table fits into the visible part of the screen

{Navigate.GoTo Up | Left | Right | Down | TopLeft | TopRight | BottomLeft | BottomRight , <Extend?(0|1)>}

Go to the sides or corners of a table. When the optional Extend? argument is 1, cell selection is extended.

{Navigate.Jump Up | Left | Right | Down}

Jump to the next table in a given direction, or jump to the current table boundary if in the middle of a table.

## {NEXTPANE}

## Syntax

{NEXTPANE <CellAtPointer?>}

## PerfectScript Syntax

NextPane ()

## Description

{NEXTPANE} switches between the panes of a notebook window previously split. 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.

## Parameters

<i>CellAtPointer?</i>	Specifies which cell should be active when the pane switches (0 or 1, optional)
-----------------------	---

## ► Related topics

## **{NEXTTOPWIN}**

### **Syntax**

{NEXTTOPWIN <Number>}

### **PerfectScript Syntax**

NextTopWin ()

### **Description**

{NEXTTOPWIN} is equivalent to the Next Window key, Ctrl+F6. It makes the next window active and moves the selector to it.

### **Parameters**

*Number*      Number of times to repeat the operation (optional)

### **▶ Related topics**

## **{NEXTWIN}**

### **Syntax**

{NEXTWIN <Number>}

### **PerfectScript Syntax**

NextWin ()

### **Description**

{NEXTWIN} is equivalent to Shift+F6. It makes the bottom window active and moves the selector to it. This macro is included for compatibility with Corel Quattro Pro for DOS.

### **Parameters**

*Number*      Number of times to repeat the operation (optional)

### **▶ Related topics**

# {Notebook}

## Syntax

{Notebook.*Property*}

## Description

{Notebook} applies formatting commands to the active notebook. The table lists the possible settings for Property. You can use {Notebook?} or {Notebook!} to display the Active Notebook dialog box. {Notebook?} lets you manipulate the dialog box, whereas {Notebook!} relies on the macro to manipulate it.

<u>Display</u>	Sets display characteristics for the active notebook
<u>Group_Mode</u>	Switches Group Mode on and off
<u>Macro_Library</u>	Searches the specified library for macros not in the active notebook
<u>Palette</u>	Determines the palette of colors displayed in the cell and sheet properties. This property is obsolete.
<u>Password</u>	Sets a password for the active notebook
<u>Password_Level</u>	Sets the level of password protection for the active notebook
<u>Recalc_Settings</u>	Controls how Corel Quattro Pro updates formula results when you change values those formulas depend on
<u>System</u>	Sets the active notebook to a system notebook
<u>Zoom_Factor</u>	Sets a Zoom Factor for an entire notebook or for individual sheets

## ► Related topics

## {Notebook.Display}

### Syntax

{Notebook.Display<Option>}

### PerfectScript Syntax

Notebook\_Display (Settings:String)

Notebook\_Display\_Objects (Mode:String)

Notebook\_Display\_Show\_HorizontalScroller (Show?:Enumeration {Yes!; No!})

Notebook\_Display\_Show\_Tabs (Show?:Enumeration {Yes!; No!})

Notebook\_Display\_Show\_VerticalScroller (Show?:Enumeration {Yes!; No!})

### Description

{Notebook.Display} is equivalent to options of the notebook property Display.

### Example

This macro command hides the vertical and horizontal scroll bars of the active notebook, reveals the sheet tabs, and shows all objects.

```
{Notebook.Display "No,No,Yes,Show All"}
```

### Options

{Notebook.Display "VertScroll, HorizScroll, Tabs, Objects"}	Sets display characteristics for the active notebook
{Notebook.Display.Objects Show All Show Outline Hide}	Specifies which parts of the notebook to display
{Notebook.Display.Show_HorizontalScroller Yes No}	Displays or hides the horizontal scroll bar
{Notebook.Display.Show_Tabs Yes No}	Displays or hides the sheet tabs
{Notebook.Display.Show_VerticalScroller Yes No}	Displays or hides the vertical scroll bar

### ► Related topics

## **{Notebook.Group\_Mode}**

### **Syntax**

{Notebook.Group\_Mode On|Off}

### **PerfectScript Syntax**

Notebook\_Group\_Mode (Mode:String)

### **Description**

{Notebook.Group\_Mode} activates or deactivates group mode.

### **▶ Related topics**

## **{Notebook.Macro\_Library}**

### **Syntax**

{Notebook.Macro\_Library Yes|No}

### **PerfectScript Syntax**

Notebook\_Macro\_Library (Enable?:Enumeration {Yes!; No!})

### **Description**

{Notebook.Macro\_Library } is equivalent to options of the notebook property Macro Library. To make the active notebook a macro library, use Yes.

### **▶ Related topics**



## {Notebook.Palette}

### Syntax

{Notebook.Palette<Option>}

### Description

{Notebook.Palette} 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. You can also edit a part individually (see the second example).

### Example

{Notebook.Palette.Color\_3 "255,0,255"} sets the third color on the notebook palette to violet (Red 255, Blue 255).

{Notebook.Palette.Color\_5.Blue "135"} sets the amount of blue in the fifth color to 135.

### Options

{Notebook.Palette Color1,Color2,...,Color16}	Determines the palette of colors displayed in the cell and sheet properties
{NoteBook.Palette.Color_n "RGB value"}	Determines the palette of colors displayed in the cell and sheet properties
{NoteBook.Palette.Color_n.R GB value}	Opens a dialog box for modifying the selected color square
{NoteBook.Palette.Color_n. RGB value }	Opens a dialog box for modifying the selected color square
{NoteBook.Palette.Color_n. RGB value }	Opens a dialog box for modifying the selected color square

#### ► Note

- This command is obsolete

#### ► Related topics

## **{Notebook.Password}**

### **Syntax**

{Notebook.Password *Password*}

### **PerfectScript Syntax**

Notebook\_Password (Password:String)

Notebook\_Password\_Level (Level:String)

### **Description**

{Notebook.Password} sets the password of the active notebook. The next save operation encrypts the file on disk.

#### ► **Note**

- Before specifying a password, set the password level using [{Notebook.Password\\_Level}](#).

#### ► **Related topics**

## **{Notebook.Password\_Level}**

### **Syntax**

{Notebook.Password\_Level None|Low|Medium|High}

### **PerfectScript Syntax**

Notebook\_Password\_Level (Level:String)

### **Description**

{Notebook.Password\_Level} sets the password level of the active notebook. If you specify a password level of Low, Medium, or High, you must also specify a password using {Notebook.Password}.

### **▶ Related topics**

## **{Notebook.Recalc\_Settings}**

### **Syntax**

```
{Notebook.Recalc_Settings "Mode, Order, Iterations, , <AuditErrors?(0|1)>"}
```

### **PerfectScript Syntax**

Notebook\_Recalc\_Settings (Settings:String)

### **Description**

{Notebook.Recalc\_Settings} is equivalent to options of the notebook property Recalc Settings. This command equivalent sets the recalculation options of the active notebook. *Mode* options are "Automatic," "Background," and "Manual." *Order* can be "Column-wise," "Row-wise," or "Natural." *Iterations* specifies the number of times formulas are recalculated before calculation is considered complete (relevant only if *Order* is changed, or if you use circular references).

To highlight the source of error for each cell containing NA or ERR in the active notebook, set the optional argument *AuditErrors?* to 1.

### **▶ Related topics**

## {Notebook.Summary}

### Syntax

{Notebook.Summary.Option}

### PerfectScript Syntax

Notebook\_Summary (Settings:String)  
Notebook\_Summary\_Author (Author:String)  
Notebook\_Summary\_Comments (Comments:String)  
Notebook\_Summary\_Keywords (Keywords:String)  
Notebook\_Summary\_Subject (Subject:String)  
Notebook\_Summary\_Title (Title:String)

### Description

{Notebook.Summary} displays summary information about the current notebook.  
You can use the following options with @COMMAND to get information about the notebook.

Notebook.Statistics.Created  
Notebook.Statistics.Directory  
Notebook.Statistics.FileName  
Notebook.Statistics.Last\_Saved  
Notebook.Statistics.Last\_Saved\_By  
Notebook.Statistics.Revision\_Number

### Example

```
@COMMAND("Notebook.Statistics.Created")
```

### Options

{Notebook.Summary.Title <i>Title</i> }	Specifies a title for the notebook
{Notebook.Summary.Subject <i>Subject</i> }	Specifies a subject for the notebook
{Notebook.Summary.Author <i>Author</i> }	Specifies an author for the notebook
{Notebook.Summary.Keyword <i>s Keywords</i> }	Specifies keywords for the notebook
{Notebook.Summary.Comme <i>nts Comments</i> }	Specifies comments for the notebook

## {Notebook.System}

### Syntax

{Notebook.System Yes|No}

### PerfectScript Syntax

Notebook\_System (Enable?:Enumeration {Yes!; No!})

### Description

{Notebook.System Yes|No} makes the active notebook a system notebook.

#### ► Related topics

## **{Notebook.Zoom\_Factor}**

### **Syntax**

{Notebook.Zoom\_Factor 10-400}

### **PerfectScript Syntax**

Notebook\_Zoom\_Factor (Factor:Numeric)

### **Description**

{Notebook.Zoom\_Factor} is equivalent to options of the notebook property Zoom Factor, which sets the zoom factor of the active notebook (from 10% to 400%). This setting is for display only and does not affect printed output.

### **▶ Related topics**

## **{NUMOFF} and {NUMON}**

### **Description**

{NUMOFF} and {NUMON} are equivalent to Num Lock off and Num Lock on, respectively.

### **▶ Related topics**





## **{OBJECTSPAGEGOTO} {OLE.Option}**

### **Description**

{OBJECTSPAGEGOTO} displays the Objects sheet of the active notebook. When the Objects sheet is active, you can use {SELECTOBJECT} to select icons, and other object commands to manipulate them.

You can use {SELECTBLOCK} to move from the Objects sheet to a spreadsheet sheet.

### **▶ Related topics**

## {OLE}

### Syntax

{OLE.Option}

### PerfectScript Syntax

OLE\_ActivateAs (ObjectType:String)

OLE\_AutomaticResize (Auto?:Enumeration {Yes!; No!})

OLE\_AutomaticUpdate (Auto?:Enumeration {Yes!; No!})

OLE\_Change\_Link (Filename:String)

OLE\_Change\_To\_Picture ()

OLE\_Convert (ObjectType:String)

OLE\_Display\_As\_Icon (Icon?:Enumeration {Yes!; No!})

OLE\_DoVerb (Action:String)

OLE\_OpenEdit ()

OLE\_Update ()

### Description

{OLE} affects the selected OLE object. The type of OLE object determines what command equivalents affect it:

OLE type	Commands
Embedded	{OLE.DoVerb}, {OLE.Convert}, {OLE.Change_To_Picture}, {OLE.DisplayAsIcon}, {OLE.ActivateAs}
Linked	{OLE.DoVerb}, {OLE.Change_Link}, {OLE.Update}, {OLE.Convert}, {OLE.Change_To_Picture}, {OLE.DisplayAsIcon}, and {OLE.ActivateAs}

### Example

This macro selects an OLE object named Embedded1, lets you edit the data (in the OLE server), then converts the object into a picture (disabling the OLE link).

```
{SELECTFLOAT Embedded1}
{OLE.DoVerb Edit}
{OLE.Change_To_Picture}
```

### Options

{OLE.ActivateAs ObjectType}	Opens the object using a different but compatible application
{OLE.AutomaticResize 0 1}	Automatically resizes the object after you edit it
{OLE.AutomaticUpdate 0 1}	Turns automatic updating on or off
{OLE.Change_Link FileName}	Switches links from one file to another
{OLE.Change_To_Picture}	Clicks to convert the embedded picture to a different type
{OLE.Convert ObjectType}	Converts the embedded information to a different type
{OLE.DisplayAsIcon 0 1}	Displays the embedded object as an icon
{OLE.DoVerb Action}	Plays, edits, or opens the object
{OLE.OpenEdit}	Opens the original application to edit the object
{OLE.Update}	Refreshes links to unopened files

## {ONERROR}

### Syntax

{ONERROR BranchLocation, <MessageLocation>, <ErrorLocation>}

## Description

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 is called, the most recent {ONERROR} replaces the previous one. If an error occurs, the {ONERROR} state is "used up" and must be redeclared. It is best to declare {ONERROR} at the very beginning of your macro, or at least before any procedure is likely to result in an error.

In general, {ONERROR} will not 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 notebook's protection property is enabled
- attempts to insert a row that would push a nonblank cell off the bottom of the notebook

## Example

The following macro uses {ONERROR} by first deliberately causing an error (trying to insert a row that would push text off the bottom of the notebook), 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
```

## Parameters

<i>BranchLocation</i>	First cell of the macro to run in case of an error
<i>MessageLocation</i>	Cell in which to store any error message (optional)
<i>n</i>	
<i>ErrorLocation</i>	Cell in which to store the address of the cell containing the macro error (optional)

## ▶ Related topics

## {OnlineService}

### Syntax

```
{OnlineService ServiceName;Arguments}
```

### PerfectScript Syntax

```
OnlineService (ServiceName:String; [Arguments:String])
```

### Description

{OnlineService} launches internet URL address from a QuickButton.

### Example

```
{OnlineService  
Internet, "http://www.corel.com/products/wordperfect/cqp8/index.htm"}
```

### Parameters

<i>ServiceName</i>	A string indicating the type of online service to use.
<i>Arguments</i>	A string indicating the command line to pass to the service.

## {OPEN}

### Syntax

```
{OPEN Filename,AccessMode}
```

### Description

{OPEN} establishes a connection to *Filename* so you can use other file-access macro commands ({READ}, {WRITE}, and so on) on it. There are five different access modes:

<b>R</b> (Read-Only)	Allows only reading from this file, ensuring no changes are made to it. {WRITE} and {WRITELN} cannot be used with a file opened as read-only.
<b>M</b> (Modify)	Opens an existing file for modification. All reading and writing commands can be used with a file opened for modification. You can also use this access mode to open a stream to a communications port (for example, "COM1") or a printer port (for example, "LPT1").
<b>W</b> (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.
<b>A</b> (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.
<b>L</b> (Locate)	Opens and closes a file to locate it.

*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.

Although {OPEN} can provide access to any type of file (including .WB3 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 is not 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 executing instructions there. {OPEN *Filename*,R}, {OPEN *Filename*,M}, and {OPEN *Filename*,A} fail if the file is not found. {OPEN *Filename*,W} fails if the supplied access path or file name is invalid. If {OPEN} fails, Quattro Pro continues executing commands in the same cell as the {OPEN} command. (See the third macro below for an example.) You can use {ONERROR} to trap some errors that occur in {OPEN}.

### Example

The following example opens the file named MYDATA.TXT for reading. If the file does not exist in the default data directory, the command fails.

```
{OPEN "MYDATA.TXT",R}
```

The next example creates a file MYDATA.TXT on drive A for writing. If there is already a file on drive A with that name, it is erased. This command will fail only if there is 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 does not exist), 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 given. 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\ does not exist."}
  {DOWN}
  {PUTCELL "Create it, then try again."}
  {DOWN}
err_loc
```

## Parameters

<i>Filename</i>	file name
<i>AccessMode</i>	R, M, W, A, or L

## {Optimizer}

### Syntax

{Optimizer.Option}

### PerfectScript Syntax

Optimizer\_Add (Constraint:Numeric; Cell:String; Operator:String; [Constant:Any])  
Optimizer\_Answer\_Reporting (Cell:String)  
Optimizer\_Auto\_Scale (Auto?:Enumeration {Yes!; No!})  
Optimizer\_Change (Constraint:Numeric; Cell:String; Operator:String; [Constant:Any])  
Optimizer\_Delete (Constraint:Numeric)  
Optimizer\_Derivatives (Derivatives:String)  
Optimizer\_Detail\_Reporting (Cell:String)  
Optimizer\_Estimates (Estimates:String)  
Optimizer\_Linear (Linear?:Enumeration {Yes!; No!})  
Optimizer\_Load\_Model ()  
Optimizer\_Max\_Iters (Iters:Numeric)  
Optimizer\_Max\_Time (Time:Numeric)  
Optimizer\_Model\_Cell (Cell:String)  
Optimizer\_Precision (Precision:Numeric)  
Optimizer\_Reset ()  
Optimizer\_Save\_Model ()  
Optimizer\_Search (Search:String)  
Optimizer\_Show\_Iters (Show?:Enumeration {Yes!; No!})  
Optimizer\_Solution\_Cell (Cell:String)  
Optimizer\_Solution\_Goal (Goal:String)  
Optimizer\_Solve ()  
Optimizer\_Target\_Value (Target:Numeric)  
Optimizer\_Tolerance (Tolerance:Numeric)  
Optimizer\_Variable\_Cells (Cell:String)

### Description

{Optimizer} performs goal-seeking calculations and solves sets of linear and nonlinear equations and inequalities.

*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}

You can use {Optimizer?} or {Optimizer!} to display the Optimizer dialog box. {Optimizer?} lets the user manipulate the dialog box, whereas {Optimizer!} relies on the macro to manipulate it.

### Example

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 selections.

```
{Optimizer.Solution_cell A:D6}  
{Optimizer.Solution_goal Max}  
{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.Estimates Quadratic}
{Optimizer.Derivatives Central}
{Optimizer.Search Conjugate}
{Optimizer.Detail_Reporting A:T2..A:T2}
{Optimizer.Answer_Reporting A:G13..A:G13}
{Optimizer.Solve}

```

## Options

{Optimizer.Add <i>Constraint#</i> , <i>Cell</i> , <= >= Integer, <i>Constant</i> }	Adds a new constraint
{Optimizer.Answer_Reporting <i>Cell</i> }	Specifies the cells for the Answer Report
{Optimizer.Auto-scale 0 1}	Automatically scales variables to achieve a target value
{Optimizer.Change <i>Constraint#</i> , <i>Cell</i> , <= >=  Integer, <i>Constant</i> }	Edits the selected constraint
{Optimizer.Delete <i>Constraint#</i> }	Removes the selected constraint
{Optimizer.Derivatives Central Forward}	Selects differencing for estimates of partial derivatives
{Optimizer.Detail_Reporting <i>Cell</i> }	Specifies the cells for the Detail Report
{Optimizer.Estimates Quadratic Tangent}	Specifies the approach used to obtain initial estimates of the basic variables in each iteration
{Optimizer.Linear 0 1}	Uses a linear method to solve the problem
{Optimizer.Load_Model}	Loads cells of Optimizer settings
{Optimizer.Max_Iters <i>Value</i> }	Sets the maximum number of iterations or trails
{Optimizer.Max_Time <i>Value</i> }	Indicates how long Optimizer can spend looking for the best solution
{Optimizer.Model_Cell <i>Cell</i> }	Saves cells of Optimizer settings for later use
{Optimizer.Precision <i>Value</i> }	Controls the accuracy of the solution
{Optimizer.Reset}	Clears Optimizer settings
{Optimizer.Save_Model}	Saves cells of Optimizer settings for future use
{Optimizer.Search Conjugate  Newton}	Selects a method for computing the search direction
{Optimizer.Show_Iters 0 1}	Pauses between iterations so you can check the progress of the search
{Optimizer.Solution_Cell <i>SolutionCell</i> }	Specifies the cell whose value you want Optimizer to measure
{Optimizer.Solution_Goal Max Min None Target <i>Value</i> }	Specifies maximum, minimum, and target values
{Optimizer.Solve}	Finds a solution to the defined problem
{Optimizer.Target_Value <i>Value</i> }	Specifies the value to be reached by the formula in the Solution Cell
{Optimizer.Tolerance <i>Value</i> }	Indicates the maximum percentage a solution can differ from a theoretical optimum integer

{Optimizer.Variable\_Cells  
Cell(s)}

solution  
Specifies the cells the Optimizer can adjust to  
reach an optimal solution

## **{Order}**

### **Syntax**

{Order.Option}

### **PerfectScript Syntax**

Order\_Backward ()

Order\_Forward ()

Order\_ToBack ()

Order\_ToFront ()

### **Description**

{Order} reorders overlapping objects in a chart or dialog window. Each command affects selected objects in the active window.

### **Options**

{Order.Backward}	Sends the selected object back one layer
{Order.Forward}	Sends the selected object forward one layer
{Order.ToBack}	Sends the selected object to the back layer
{Order.ToFront}	Sends the selected object to the front layer

### **▶ Related topics**



# {Outline}

## Syntax

{Outline.Option}

## PerfectScript Syntax

Outline\_AutoOutline ()

Outline\_Collapse ()

Outline\_Expand ()

Outline\_Group ()

Outline\_Hide (Hide?:Numeric)

Outline\_Summary (Row?:Enumeration {Above!; Below!}; Col?:Enumeration {Left!; Right!})

Outline\_ToLevel (RowCol?:String; [Level?:Numeric])

Outline\_Ungroup ()

Outline\_UnGroupAll ()

## Description

{Outline} defines, creates, manipulates, and groups outlines.

## Options

{Outline.AutoOutline}	Creates an outline automatically on the current pane/page
{Outline.Group}	Groups rows or columns. If the cells are not an entire row or column, whichever one contains the most elements (rows or columns) will be grouped
{Outline.Ungroup}	Ungroups rows or columns. If the cells are not an entire row or column, whichever one contains the most elements (rows or columns) will be ungrouped. If the cells do not span the ENTIRE group, only those rows/columns that are inside the cells will be ungrouped
{Outline.UngroupAll}	Destroys all groups on the current pane/page
{Outline.Expand}	Expands a collapsed group of rows or columns. If the cells are not an entire row or column and are inside a group, whichever one contains the most elements (rows or columns) and is inside a current group, will be expanded
{Outline.Collapse}	Collapses an expanded group of rows or columns. If the cells are not an entire row or column and are inside a group, whichever one contains the most elements (rows or columns) and is inside a current group will be collapsed
{Outline.Hide 0 1}	Either hides or shows the outline in the current pane/page
{Outline.Summary Above Below, Left Right}	Sets whether the summary will be above or below for row-based groups, and left or right for column-based groups
{Outline.ToLevel Rows Columns, Level}	Collapses or expands a group or rows or columns at a specific level



## {Page}

### Syntax

{Page.*Property*}

### Description

{Page} affects the active sheet(s). The next table lists the possible settings for *Property*. To display a property description with syntax, choose the property in the following list:

<b>Property</b>	<b>Description</b>
<u>Conditional_Color</u>	Changes the color of specific types of data in the active sheet: values above or below a specified range, and ERR values
<u>Default_Width</u>	Sets the default width of all columns in the active sheet
<u>Display</u>	Sets display characteristics for the active sheet
<u>Name</u>	Controls the name of the active sheet
<u>Protection</u>	Turns on protection in the active sheet
<u>Tab_Color</u>	Changes the tab color of the active sheet
<u>Zoom_Factor</u>	Lets you pull back to see a whole printed page, or focus in on the detail of a few cells

You can use {Page?} or {Page!} to display the Active Sheet dialog box. {Page?} lets you manipulate the dialog box, whereas {Page!} relies on the macro to manipulate it.

#### ► Related topics

# {Page.Conditional\_Color}

## Syntax

{Page.Conditional\_Color<Option>}

## PerfectScript Syntax

Page\_Conditional\_Color (Settings:String)  
Page\_Conditional\_Color\_Above\_Normal\_Color (ColorID:Numeric)  
Page\_Conditional\_Color\_Below\_Normal\_Color (ColorID:Numeric)  
Page\_Conditional\_Color\_Enable (Enable?:Enumeration {Yes!; No!})  
Page\_Conditional\_Color\_ERR\_Color (ColorID:Numeric)  
Page\_Conditional\_Color\_Greatest\_Normal\_Value (Value:Numeric)  
Page\_Conditional\_Color\_Normal\_Color (ColorID:Numeric)  
Page\_Conditional\_Color\_Smallest\_Normal\_Value (Value:Numeric)

## Description

{Page.Conditional\_Color} is equivalent to the sheet 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).

## Example

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"}
```

## Options

{Page.Conditional\_Color "*Enable, SmallVal, GreatVal, BelowColor, NormalColor, AboveColor, ERRColor*"}

Changes the color of specific types of data in the active sheet: values above or below a specified range, and ERR values

{Page.Conditional\_Color.Above\_Normal\_Color 0-15}

Sets the color of cells whose values are above the Greatest Normal Value

{Page.Conditional\_Color.Below\_Normal\_Color 0-15}

Sets the color of cells whose values are below the Smallest Normal Value

{Page.Conditional\_Color.Enable Yes|No}

Indicates whether to use the conditional colors set with this property

{Page.Conditional\_Color.ERR\_Color 0-15}

Specifies the color to use for ERR and NA values generated by formula errors

{Page.Conditional\_Color.Greatest\_Normal\_Value *Value*}

Specifies the largest value of the range of values you consider normal

{Page.Conditional\_Color.Normal\_Color 0-15}

Sets the color of cells whose value falls within the range set by the Smallest Normal Value and the Greatest Normal Value

{Page.Conditional\_Color.Smallest\_Normal\_Value *Value*}

Specifies the smallest value of the range of values you consider normal

## ► Related topics

## **{Page.Default\_Width}**

### **Syntax**

{Page.Default\_Width *Width*}

### **PerfectScript Syntax**

Page\_Default\_Width (Width:Numeric)

### **Description**

{Page.Default\_Width} is equivalent to the sheet property Default Width. It sets the default column width of the active sheet. *Width* is the new column width in twips (a twip is 1/1440th of an inch).

### **Example**

{Page.Default\_Width "720"} makes the default column width a half inch (720 twips).

### **▶ Related topics**

# {Page.Display}

## Syntax

{Page.Display<*Option*>}

## PerfectScript Syntax

Page\_Display (Settings:String)

Page\_Display\_Borders (Settings:String)

Page\_Display\_Borders\_Column\_Borders (Show?:Enumeration {Yes!; No!})

Page\_Display\_Borders\_Row\_Borders (Show?:Enumeration {Yes!; No!})

Page\_Display\_Display\_Zeros (Show?:Enumeration {Yes!; No!})

Page\_Display\_Grid\_Lines (Settings:String)

Page\_Display\_Grid\_Lines\_Horizontal (Show?:Enumeration {Yes!; No!})

Page\_Display\_Grid\_Lines\_Vertical (Show?:Enumeration {Yes!; No!})

## Description

{Page.Display} is equivalent to the sheet property Display, which sets the display of zeros, borders, and grid lines. The arguments of {Page.Display} (which sets all options of the Display property in one command) use the same syntax as those in the {Page.Display.*Option*} commands. All {Page.Display} arguments take Yes|No string values.

## Example

The following macro displays zero values on the sheet, but hides borders and grid lines.

```
{Page.Display "Yes,No,No,No,No,No"}
```

## Options

{Page.Display <i>DisplayZeros?</i> (Yes No), <i>RowBorders?</i> (Yes No), <i>ColumnBorders?</i> (Yes No), <i>HorzGridLines?</i> (Yes No), <i>VertGridLines?</i> (Yes No)}	Sets display characteristics for the active sheet
{Page.Display.Borders " <i>RowBorders?</i> (Yes No), <i>ColumnBorders?</i> (Yes No)}	Turns border options off and on in the active sheet
{Page.Display.Borders.Column_Borders Yes No}	Turns column borders off and on in the active sheet
{Page.Display.Borders.Row_Borders Yes No}	Turns row borders off and on in the active sheet
{Page.Display.Display_Zeros Yes No}	Suppresses display of any value in the active sheet that exactly equals zero
{Page.Display.Grid_Lines " <i>HorizGridLines?</i> (Yes No), <i>VertGridLines?</i> (Yes No)"}	Turns spreadsheet grid off and on in the active sheet
{Page.Display.Grid_Lines.Horizontal Yes No}	Turns horizontal spreadsheet grid off and on in the active sheet
{Page.Display.Grid_Lines.Vertical Yes No}	Turns vertical spreadsheet grid off and on in the active sheet

## ► Related topics

## **{Page.Name}**

### **Syntax**

{Page.Name *NewName*}

### **PerfectScript Syntax**

Page\_Name (*NewName*:String)

### **Description**

{Page.Name *NewName*} is equivalent to the sheet property Name. It sets the name of the active sheet to *NewName*.

### **▶ Related topics**

## **{Page.Protection}**

### **Syntax**

{Page.Protection<*Option*>}

### **Syntax**

Page\_Protection (Settings:String)

Page\_Protection\_Cells (Protect?:Enumeration {Yes!; No!})

Page\_Protection\_Objects (Protect?:Enumeration {Yes!; No!})

### **Description**

{Page.Protection} is equivalent to the sheet property Protection. It enables or disables cell and object protection on the active sheet.

### **Options**

{Page.Protection "CellLocking?(Yes No), ObjectLocking?(Yes  No)"}	Turns on protection in the active sheet
{Page.Protection.Cells Yes No}	Protects all cell entries in the active sheet
{Page.Protection.Objec ts Yes No}	Protects all objects in the active sheet

### **▶ Related topics**



## **{Page.Tab\_Color}**

### **Syntax**

{Page.Tab\_Color "Red, Green, Blue, UseRGB?"}

### **PerfectScript Syntax**

Page\_Tab\_Color (Settings:String)

### **Description**

{Page.Tab\_Color} changes the tab color of the active sheet; *Red*, *Green*, and *Blue* are integers from 0 to 255.

### **▶ Related topics**

## **{Page.Zoom\_Factor}**

### **Syntax**

{Page.Zoom\_Factor 10-400}

### **PerfectScript Syntax**

Page\_Zoom\_Factor (Factor:Numeric)

### **Description**

{Page.Zoom\_Factor} sets the zoom factor of the active sheet (from 10% to 400%). This setting is for display only and does not affect printed output.

### **▶ Related topics**

## **{PageViewGoto}**

### **Description**

Switches from either the Objects Sheet or the sheet in Draft mode to Page View.

### **▶ Related topics**

## **{PANELOFF}**

### **Description**

{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 significantly 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 executing, so you need not worry about locking macro users out of the menus. To cancel its effect during macro execution, use [{PANELON}](#).

{PANELOFF} does not 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.

### **▶ [Related topics](#)**

## **{PANELON}**

### **Description**

{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.

### **▶ Related topics**

## **{ParseExpert.ApplyFormatting}**

### **Syntax**

{ParseExpert.ApplyFormatting *Apply*}

### **PerfectScript Syntax**

ParseExpert\_ApplyFormatting (Apply?:Enumeration {Yes!; No!})

### **Description**

Lets you specify whether the column alignment and format specified in the Preview pane should be applied to the destination cells.

### **Parameter**

<i>Apply</i>	0 Do not apply to the destination cells. 1 Apply to the destination cells.
--------------	---

## **{ParseExpert.CellDelimiterString}**

### **Syntax**

{ParseExpert.CellDelimiterString *Value*}

### **PerfectScript Syntax**

ParseExpert\_CellDelimiterString (Value?: String)

### **Description**

Lets you specify the string to use as the cell delimiter.

### **Parameter**

<i>Value</i>	The string
--------------	------------

## **{ParseExpert.CellDelimiterTypeComma}**

### **Syntax**

{ParseExpert.CellDelimiterTypeComma *Enable*}

### **PerfectScript Syntax**

ParseExpert\_CellDelimiterTypeComma {Yes!; No!}

### **Description**

Lets you specify whether or not to make the cell delimiter a comma.

### **Parameter**

<i>Enable</i>	0 Do not make the cell delimiter a comma. 1 Make the cell delimiter a comma
---------------	--

## **{ParseExpert.CellDelimiterTypeMultiSpace}**

### **Syntax**

{ParseExpert.CellDelimiterTypeMultiSpace *Enable*}

### **PerfectScript Syntax**

ParseExpert\_CellDelimiterTypeMultiSpace {Yes!; No!}

### **Description**

Lets you specify whether or not to make the cell delimiter a multi-space.

## Parameter

*Enable*

0 Do not make the cell delimiter a multi-space.  
1 Make the cell delimiter a multi-space.

## {ParseExpert.CellDelimiterTypeOther}

### Syntax

{ParseExpert.CellDelimiterTypeOther *Enable*}

### PerfectScript Syntax

ParseExpert\_CellDelimiterTypeOther {Yes!; No!}

### Description

Lets you specify whether or not to make the cell delimiter a character other than a comma, a multi-space, a semicolon, a space, or a tab.

## Parameter

*Enable*

0 Do not make the cell delimiter a character other than a comma, a multi-space, a semicolon, a space, or a tab.  
1 Make the cell delimiter a character other than a comma, a multi-space, a semicolon, a space, or a tab.

## {ParseExpert.CellDelimiterTypeReturn}

### Syntax

{ParseExpert.Return *Enable*}

### PerfectScript Syntax

ParseExpert\_CellDelimiterTypeReturn {Yes!; No!}

### Description

Lets you specify whether or not to make the cell delimiter a carriage return.

## Parameter

*Enable*

0 Do not make the cell delimiter a carriage return.  
1 Make the cell delimiter a carriage return.

## {ParseExpert.CellDelimiterTypeSemiColon}

### Syntax

{ParseExpert.CellDelimiterTypeSemiColon *Enable*}

### PerfectScript Syntax

ParseExpert\_CellDelimiterTypeSemiColon {Yes!; No!}

### Description

Lets you specify whether or not to make the cell delimiter a semicolon.

## Parameter

*Enable*

0 Do not make the cell delimiter a semicolon.  
1 Make the cell delimiter a semicolon.

## **{ParseExpert.CellDelimiterTypeSpace}**

### **Syntax**

{ParseExpert.CellDelimiterTypeSpace *Enable*}

### **PerfectScript Syntax**

ParseExpert\_CellDelimiterTypeSpace {Yes!; No!}

### **Description**

Lets you specify whether or not to make the cell delimiter a space.

### **Parameter**

*Enable*

0 Do not make the cell delimiter a space.  
1 Make the cell delimiter a space.

## **{ParseExpert.CellDelimiterTypeTab}**

### **Syntax**

{ParseExpert.CellDelimeterTypeTab *Enable*}

### **PerfectScript Syntax**

ParseExpert\_CellDelimiterTypeTab {Yes!; No!}

### **Description**

Lets you specify whether or not to make the cell delimiter a tab.

### **Parameter**

*Enable*

0 Do not make the cell delimiter a tab.  
1 Make the cell delimiter a tab.

## **{ParseExpert.ColumnWidths}**

### **Syntax**

{ParseExpert.ColumnWidths *Apply*}

### **PerfectScript Syntax**

ParseExpert\_ColumnWidths (Apply?:Enumeration {Yes!; No!})

### **Description**

Lets you specify whether or not the columns widths specified in the preview pane should be applied to the destination cells.

### **Parameter**

*Apply*

0 Do not apply to the destination cells.  
1 Apply to the destination cells.

## **{ParseExpert.ConsecutiveAsOne}**

### **Syntax**

{ParseExpert.ConsecutiveAsOne *Apply*}

### **PerfectScript Syntax**

ParseExpert\_ConsecutiveAsOne (Apply?:Enumeration {Yes!; No!})

### **Description**

Lets you specify whether or not to skip the delimiters that do not enclose data.

### **Parameter**



*Apply*

0 Do not skip the delimiters  
1 Skip the delimiters

## **{ParseExpert.DataType}**

### **Syntax**

{ParseExpert.DataType *Type*}

### **PerfectScript Syntax**

ParseExpert\_DataType (*Type?:String*)

### **Description**

Lets you specify which additional parse options are displayed.

### **Parameter**

*Type*

"Fixed" Display the fixed parse options.  
"Delimited" Display the delimited parse options.

## **{ParseExpert.DelimiterType}**

### **Syntax**

{ParseExpert.DelimiterType *Type*}

### **PerfectScript Syntax**

ParseExpert\_DelimiterType (*Type?:String*)

### **Description**

Lets you specify which delimiter separates text.

### **Parameter**

*Type*

"Space" Separates text with a space.  
"Tab" Separates text with a tab.  
"Comma" Separates text with a comma.  
"CommaQuote" Separates text with a comma quote.  
"Other" Separates text with a delimiter other than a space, a tab, a comma, or a comma quote.

## **{ParseExpert.Go}**

### **Syntax**

{ParseExpert.Go}

### **PerfectScript Syntax**

ParseExpert\_Go ()

### **Description**

Parses the text and copies it as data to the destination cells.

## **{ParseExpert.IgnoreNonConformingRows}**

### **Syntax**

{ParseExpert.IgnoreNonConformingRows *Apply*}

### **PerfectScript Syntax**

ParseExpert\_IgnoreNonConformingRows (Apply?:Enumeration {Yes!; No!})

### **Description**

Lets you specify whether or not to skip the lines in the text that the QuickColumns Expert cannot parse.

### **Parameter**

<i>Apply</i>	0 Do not skip the lines. 1 Skip the lines.
--------------	---

## **{ParseExpert.InputBlock}**

### **Syntax**

{ParseExpert.InputBlock *Block*}

### **PerfectScript Syntax**

ParseExpert\_InputBlock (Block?:String)

### **Description**

Lets you specify the range of cells to parse.

### **Parameter**

<i>Block</i>	The range of cells
--------------	--------------------

## **{ParseExpert.InputFile}**

### **Syntax**

{ParseExpert.InputFile *Filename*}

### **PerfectScript Syntax**

ParseExpert\_InputFile (Filename?:String)

### **Description**

Lets you specify the name of the file.

### **Parameter**

<i>Filename</i>	The name of the file
-----------------	----------------------

## **{ParseExpert.InputType}**

### **Syntax**

{ParseExpert.InputType *Type*}

### **PerfectScript Syntax**

ParseExpert\_InputType (Type?:String)

### **Description**

Lets you specify whether you want to parse data from a file or from the spreadsheet.

### **Example**

```
{ParseExpert.InputType "Block"}
```

Result: Parse data from the spreadsheet.

## Parameter

*Type*

File

Parse data from a file.

Block

Parse data from the spreadsheet.

## {ParseExpert.JoinBrokenLines}

### Syntax

{ParseExpert.JoinBrokenLines *Apply*}

### PerfectScript Syntax

ParseExpert\_JoinBrokenLines (Apply?:Enumeration {Yes!; No!})

### Description

Lets you specify whether or not to restore the wrapped lines in the text file to single lines.

### Parameter

*Apply*

0 Do not restore the wrapped lines.

1 Restore the wrapped lines.

## {ParseExpert.LineLength}

### Syntax

{ParseExpert.LineLength *Length*}

### PerfectScript Syntax

ParseExpert\_LineLength (Length?:Numeric)

### Description

Lets you specify the number of characters to count before restoring wrapped lines to single files.

### Parameter

*Length*

The number of characters to count

## {ParseExpert.LoadSettings}

### Syntax

{ParseExpert.LoadSettings}

### PerfectScript Syntax

ParseExpert\_LoadSettings ()

### Description

Loads the saved parse settings.

## **{ParseExpert.OtherDelimiter}**

### **Syntax**

{ParseExpert.OtherDelimiter *Delimiter*}

### **PerfectScript Syntax**

ParseExpert\_OtherDelimiter (Delimiter?:String)

### **Description**

Lets you specify the character to separate the text other than a tab, a comma, a quote, or a space.

### **Parameter**

*Delimiter* The character to separate the text

## **{ParseExpert.OutputBlock}**

### **Syntax**

{ParseExpert.OutputBlock *Block*}

### **PerfectScript Syntax**

ParseExpert\_OutputBlock (Block?:String)

### **Description**

Lets you specify the cells where you want to enter the parsed text.

### **Parameter**

*Block* The cells where you want to enter the parsed text

## **{ParseExpert.PageLength}**

### **Syntax**

{ParseExpert.PageLength *Length*}

### **PerfectScript Syntax**

ParseExpert\_PageLength (Length?:Numeric)

### **Description**

Lets you specify the number of unparsed text lines on each page.

### **Parameter**

*Length* The number of unparsed text lines

## **{ParseExpert.PageLengthEnabled}**

### **Syntax**

{ParseExpert.PageLengthEnabled *Apply*}

### **PerfectScript Syntax**

ParseExpert\_PageLengthEnabled (Apply?:Enumeration {Yes!; No!})

### **Description**

Lets you specify whether to skip text rows or to copy text rows into the destination cells as unparsed text.

### **Parameter**

*Apply* 0 Skips text rows  
1 Copies text rows

## **{ParseExpert.Restore}**

### **Syntax**

{ParseExpert.Restore}

### **PerfectScript Syntax**

ParseExpert\_Restore ()

### **Description**

Restores the current page settings to the default page settings.

#### **► Note**

- You do not need to use this command in versions of Quattro Pro later than Corel Quattro Pro 8.

## **{ParseExpert.RowDelimiterString}**

### **Syntax**

{ParseExpert.RowDelimiterString *Value*}

### **PerfectScript Syntax**

ParseExpert\_RowDelimiterString (Value?:String)

### **Description**

Lets you specify the row delimiter

### **Parameter**

*Value* The row delimiter

## **{ParseExpert.RowDelimiterTypeComma}**

### **Syntax**

{ParseExpert.RowDelimiterTypeComma *Enable*}

### **PerfectScript Syntax**

ParseExpert\_RowDelimiterTypeComma {Yes!; No!}

### **Description**

Lets you specify whether or not to make the row delimiter a comma.

### **Parameter**

*Enable* 0 Do not make the row delimiter a comma.  
1 Make the row delimiter a comma.

## **{ParseExpert.RowDelimiterTypeMultiSpace}**

### **Syntax**

{ParseExpert.RowDelimiterTypeMultiSpace *Enable*}

### **PerfectScript Syntax**

ParseExpert\_RowDelimiterTypeMultiSpace {Yes!; No!}

### **Description**

Lets you specify whether or not to make the row delimiter a multi-space.

### **Parameter**

*Enable* 0 Do not make the row delimiter a multi-space  
1 Make the row delimiter a multi-space.

## **{ParseExpert.RowDelimiterTypeOther}**

### **Syntax**

{ParseExpert.RowDelimiterTypeOther *Enable*}

### **PerfectScript Syntax**

ParseExpert\_RowDelimiterTypeOther {Yes!; No!}

### **Description**

Lets you specify whether or not to make the row delimiter a character other than a comma, a multi-space, a semicolon, a space, or a tab.

## Parameter

*Enable*

0 Do not make the row delimiter a character other than a comma, a multi-space, a semicolon, a space, or a tab.  
1 Make the row delimiter a character other than a comma, a multi-space, a semicolon, a space, or a tab.

## {ParseExpert.RowDelimiterTypeReturn}

### Syntax

{ParseExpert.RowDelimiterTypeReturn *Enable*}

### PerfectScript Syntax

ParseExpert\_RowDelimiterTypeReturn {Yes!; No!}

### Description

Lets you specify whether or not to make the row delimiter a carriage return.

## Parameter

*Enable*

0 Do not make the row delimiter a carriage return.  
1 Make the row delimiter a carriage return.

## {ParseExpert.RowDelimiterTypeSemiColon}

### Syntax

{ParseExpert.RowDelimiterTypeSemiColon *Enable*}

### PerfectScript Syntax

ParseExpert\_RowDelimiterTypeSemiColon {Yes!; No!}

### Description

Lets you specify whether or not to make the row delimiter a semicolon.

## Parameter

*Enable*

0 Do not make the row delimiter a semicolon.  
1 Make the row delimiter a semicolon.

## {ParseExpert.RowDelimiterTypeSpace}

### Syntax

{ParseExpert.RowDelimiterTypeSpace *Enable*}

### PerfectScript Syntax

ParseExpert\_RowDelimiterTypeSpace {Yes!; No!}

### Description

Lets you specify whether or not to make the row delimiter a space.

## Parameter

*Enable*

0 Do not make the row delimiter a space.  
1 Make the row delimiter a space.

## {ParseExpert.RowDelimiterTypeTab}

## Syntax

{ParseExpert.RowDelimiterTypeTab *Enable*}

## PerfectScript Syntax

ParseExpert\_RowDelimiterTypeTab {Yes!; No!}

## Description

Lets you specify whether or not to make the row delimiter a tab.

## Parameter

*Enable*

0 Do not make the row delimiter a tab.

1 Make the row delimiter a tab.

## {ParseExpert.SaveSettings}

### Syntax

{ParseExpert.SaveSettings}

### PerfectScript Syntax

ParseExpert\_SaveSettings ()

### Description

Saves the current parse settings.



## **{ParseExpert.SettingsFile}**

### **Syntax**

{ParseExpert.SettingsFile *Filename*}

### **PerfectScript Syntax**

ParseExpert\_SettingsFile (Filename?:String)

### **Description**

Save the current parse settings as a file.

### **Parameter**

*Filename* The name of the file

## **{ParseExpert.SheetDelimiterString}**

### **Syntax**

{ParseExpert.SheetDelimiterString *Value*}

### **PerfectScript Syntax**

ParseExpert\_SheetDelimiterString (Value?: String)

### **Description**

Lets you specify the sheet delimiter.

### **Parameter**

*Value* The sheet delimiter

## **{ParseExpert.SheetDelimiterTypeComma}**

### **Syntax**

{ParseExpert.SheetDelimiterTypeComma *Enable*}

### **PerfectScript Syntax**

ParseExpert\_SheetDelimiterTypeComma (Yes!; No!)

### **Description**

Lets you specify whether or not to make the sheet delimiter a comma.

### **Parameter**

*Enable* 0 Do not make the sheet delimiter a comma.  
1 Make the sheet delimiter a comma.

## **{ParseExpert.SheetDelimiterTypeMultiSpace}**

### **Syntax**

{ParseExpert.SheetDelimiterTypeMultiSpace *Enable*}

### **PerfectScript Syntax**

ParseExpert\_SheetDelimiterTypeMultiSpace (Yes!; No!)

### **Description**

Lets you specify whether or not to make the sheet delimiter a multi-space.

### **Parameter**

*Enable* 0 Do not make the sheet delimiter a

multi-space.  
1 Make the sheet delimiter a multi-space.

## **{ParseExpert.SheetDelimiterTypeOther}**

### **Syntax**

{ParseExpert.SheetDelimiterTypeOther *Enable*}

### **PerfectScript Syntax**

ParseExpert\_SheetDelimiterTypeOther (Yes!; No!)

### **Description**

Lets you specify whether or not to make the sheet delimiter a character other than a comma, a multi-space, a semicolon, a space, or a tab.

### **Parameter**

*Enable*

0 Do not make the sheetdelimiter a character other than a comma, a multi-space, a semicolon, a space, or a tab.  
1 Make the sheet delimiter a character other than a comma, a multi-space, a semicolon, a space, or a tab.

## **{ParseExpert.SheetDelimiterTypeReturn}**

### **Syntax**

{ParseExpert.SheetDelimiterTypeReturn *Enable*}

### **PerfectScript Syntax**

ParseExpert\_SheetDelimiterTypeReturn (Yes!; No!)

### **Description**

Lets you specify whether or not to make the sheet delimiter a carriage return.

### **Parameter**

*Enable*

0 Do not make the sheet delimiter a carriage return.  
1 Make the sheet delimiter a carriage return.

## **{ParseExpert.SheetDelimiterTypeSemiColon}**

### **Syntax**

{ParseExpert.SheetDelimiterTypeSemiColon *Enable*}

### **PerfectScript Syntax**

ParseExpert\_SheetDelimiterTypeSemiColon (Yes!; No!)

### **Description**

Lets you specify whether or not to make the sheet delimiter a semicolon.

### **Parameter**

*Enable*

0 Do not make the sheet delimiter a semicolon.  
1 Make the sheet delimiter a semicolon.

## **{ParseExpert.SheetDelimiterTypeSpace}**

## Syntax

{ParseExpert.SheetDelimiterTypeSpace *Enable*}

## PerfectScript Syntax

ParseExpert\_SheetDelimiterTypeSpace (Yes!; No!)

## Description

Lets you specify whether or not to make the sheet delimiter a space.

## Parameter

*Enable*

0 Do not make the sheet delimiter a space.

1 Make the sheet delimiter a space.

## {ParseExpert.SheetDelimiterTypeTab}

### Syntax

{ParseExpert.SheetDelimiterTypeTab *Enable*}

### PerfectScript Syntax

ParseExpert\_SheetDelimiterTypeTab (Yes!; No!)

### Description

Lets you specify whether or not to make the sheet delimiter a tab.

### Parameter

*Enable*

0 Do not make the sheet delimiter a tab.

1 Make the sheet delimiter a tab.

## {ParseExpert.Skip1stChar}

### Syntax

{ParseExpert.Skip1stChar *Apply*}

### PerfectScript Syntax

ParseExpert\_Skip1stChar (Apply?: Enumeration {Yes!; No!})

### Description

Lets you specify whether or not to skip the first character in a line of text.

### Parameter

*Apply*

0 Do not skip the first character.

1 Skip the first character.

## {ParseExpert.TextQualifier}

### Syntax

{ParseExpert.TextQualifier *Type*}

### PerfectScript Syntax

ParseExpert\_TextQualifier (Type: String)

### Description

Lets you specify the character that appears before and after any instance of data that contains the character specified by Other.

### Parameter

*Type*

"SingleQuote"

"DoubleQuote"  
"None"

## {ParseExpert.ValueQualifier}

### Syntax

{ParseExpert.ValueQualifier *Type*}

### PerfectScript Syntax

ParseExpert\_ValueQualifier (*Type?*: String)

### Description

Lets you specify the character that appears before and after any instance of data that should be parsed as a value.

### Parameter

*Type* "SingleQuote"  
"DoubleQuote"  
"None"

## {Parse}

### Syntax

{Parse.*Option*}

### PerfectScript Syntax

Parse\_Create ()  
Parse\_EditLine ()  
Parse\_Go ()  
Parse\_Input (Block:String)  
Parse\_Output (Block:String)  
Parse\_Reset ()

### Description

{Parse} breaks long text strings into data fields according to a format line. {Parse.Create} and {Parse.EditLine} act on the active cells.

### Example

The macro in the next figure selects text cells, builds a format line in the first line of the selected cells, identifies the format line and text as the parse input cells, specifies output cells, and performs the parse:

	A	B	C	D	E	
6	V>>>*L>>>>>>> *L>>>*****L> >>>>>>>>>*L					{SelectBlock A:A6..A:D7}
7	1968,Davis,John, Harisburg,PA					{Parse.Input A:A6..A:D7}
8	1972,Lee,Elizabe th,New York,NY					{Parse.Output A:A10}
9						{Parse.Create}
10						{Parse.Input A:A7..A:D8}
11	1968	Davis	John	Harrisbu	PA	{Parse.Output A:A11}
11		,		rg,		
11	1972	Lee,	Elizabet	New	NY	{Parse.Go}
12			h	York,		

To use {Parse.EditLine}, follow the macro with a {CLEAR} macro command, then the string for the new format line, and a {CR} macro, as shown in the following example:

```
{Parse.EditLine}{CLEAR}V>>>*L>>>>>>>>>*****L>>>>>>>>>*L{CR}
```

## Options

{Parse.Create}	Use to build the format line.
{Parse.EditLine}	Lets you specify a new format line.
{Parse.Go}	Use after setting up input and output cells and creating a format line.
{Parse.Input Block}	Indicates the cells to parse.
{Parse.Output Block}	Indicates the cells to hold parsed data.
{Parse.Reset}	Clears previous settings.

## {PasteFormat}

### Syntax

{PasteFormat *LinkType*}

### PerfectScript Syntax

PasteFormat (LinkType:String)

### Description

{PasteFormat} lets you paste data in a specific format (for example, an OLE object) into a notebook. Use *LinkType* to specify the paste format.

### Example

{PasteFormat Bitmap} pastes the data in the Clipboard as a bitmap into the active notebook.

You can use {PasteFormat?} or {PasteFormat!} to display the Paste Special dialog box. {PasteSpecial?} lets you manipulate the dialog box, whereas {PasteSpecial!} relies on the macro to manipulate it.

### Parameters

*LinkType*      Format to paste object as

### ► Related topics

## **{PasteLink}**

### **Description**

{PasteLink} sets up a DDE link to another application.

► **Related topics**

# {PasteSpecial}

## Syntax

{PasteSpecial "Properties", "Formula Cells", "Label cells", "Number cells", "Formula Values", "Transpose", "NoBlanks", Cell\_Comments}

## PerfectScript Syntax

PasteSpecial ([Properties:String]; [FormulaCells:String]; [LabelCells:String]; [NumberCells:String]; [FormulaValues:String]; [Transpose:String]; [NoBlanks:String])

## Description

{PasteSpecial} pastes certain aspects of Quattro Pro data from the Clipboard.

You can use {PasteSpecial?} or {PasteSpecial!} to display the Paste Special dialog box. {PasteSpecial?} lets you manipulate the dialog box, whereas {PasteSpecial!} relies on the macro to manipulate it.

## Example

The following macro pastes properties, formula cells, and numbers from the Clipboard, and skips any blank cells.

```
{PasteSpecial Properties, Formula Cells,"",Number cells,"","",NoBlanks,""}
```

## Parameters

Properties	Properties to paste from Clipboard; "" otherwise
Formula Cells	Formula cells to paste from Clipboard, "" otherwise
Number Cells	Number cells to paste from Clipboard, "" otherwise
Formula Values	Pastes formula cells as values, "" otherwise
Transpose	Switches the position of entries (data listed in columns is placed in rows and vice versa), "" otherwise
NoBlanks	Avoids pasting blank cells from Clipboard; "" otherwise
Cell_Comments	Pastes cell comments; "" otherwise

## ▶ Related topics

## **{PAUSEMACRO}**

### **Description**

{PAUSEMACRO} is used with [{DODIALOG}](#) or a command equivalent invoked with ! to pause the macro so that you can "finish up" whatever dialog box is displaying. Once you finish using the dialog box (by choosing OK, or canceling it), macro execution resumes with any macro commands following the {PAUSEMACRO}.

Use {PAUSEMACRO} only when a dialog box is displaying, Quattro Pro is in FIND mode, or Quattro Pro is in INPUT mode (using {RestrictInput.*Option*}). Otherwise, the macro pauses indefinitely.

### **Example**

The following macro displays the Copy Cells dialog box, sets the From edit field to A1, activates the To edit field, and then waits for you to complete the copy operation. Once you finish the dialog box, the macro beeps and moves down a cell.

```
_copy_a1      {BlockCopy!}  
              {ALT+F}  
              A1  
              {ALT+T}  
              {PAUSEMACRO}  
              {BEEP} {DOWN}
```

### **► Related topics**



## **{PGDN} and {PGUP}**

### **Syntax**

{PGDN <Number>},{PGUP <Number>}

### **Description**

{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.

### **Parameters**

*Number* Any positive integer or address of a cell containing a positive integer (optional)

### **▶ Related topics**

## **{PlayPerfectScript}**

### **Syntax**

{PlayPerfectScript *Filename*}

### **Description**

{PlayPerfectScript} plays a PerfectScript macro you specify.

### **Parameters**

*Filename*      Name of a PerfectScript macro file to run

### **▶ Related topics**

## {POKE}

### Syntax

{POKE *DDEChannel*, *Destination*, *DataToSend*}

### PerfectScript Syntax

Poke (DDEChannel:Numeric; Destination:String; DataToSend:String)

### Description

{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; the destination could be cells in Excel or a bookmark in Word for Windows. *DataToSend* refers to cells 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*).

### Example

This 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. The command block contains an ObjectVision command not available in Quattro Pro:

```
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}
```

### Parameters

<i>DDEChannel</i>	Channel ID number of the application to send information to
<i>Destination</i>	Location in the application that receives the information being sent
<i>DataToSend</i>	Cells containing the information to send to the application

### ► Related topics

## **{Preview}**

### **Description**

{Preview} lets you preview a printout on screen.

▶ **Related topics**

## **{Print}**

### **Syntax**

{Print.*Option*}

### **Description**

{Print} is equivalent to the menu items in the following list. To display specific command equivalents, choose one of the following:

### **Command options for...**

Page Setup

Named Settings

Print

Page Setup Options

The command equivalent {Print.PrintReset} resets print settings in all the dialog boxes displayed by these commands.

You can use {Print?} or {Print!} to display the Spreadsheet Print dialog box. {Print?} lets you manipulate the dialog box, whereas {Print!} relies on the macro to manipulate it.

## Named Settings Command Options

### PerfectScript Syntax

Print\_Create (NamedSetting:String)

Print\_Delete (NamedSetting:String)

Print\_Use (NamedSetting:String)

### Description

These command options affect named settings for printing. 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.

{Print.Create <i>NamedSetting</i> }	Creates a named print setting using the name in the New Set text box
{Print.Delete <i>NamedSetting</i> }	Replaces the settings stored under the selected name with the current print settings
{Print.Delete <i>NamedSetting</i> }	Deletes the selected named setting
{Print.Use <i>NamedSetting</i> }	Uses the selected named print setting

### ► Related topics

## Page Setup Command Options

### PerfectScript Syntax

Print\_Options ()  
Print\_Bottom\_Margin (Margin:String)  
Print\_Create\_Footer (CreateFooter:Enumeration {Yes!; No!})  
Print\_Create\_Header (CreateHeader:Enumeration {Yes!; No!})  
Print\_Footer (String:String)  
Print\_Footer\_Margin (Margin:String)  
Print\_Footers\_Font (Settings:String)  
Print\_Header (String:String)  
Print\_Header\_Margin (Margin:String)  
Print\_Headers\_Font (Settings:String)  
Print\_Left\_Margin (Margin:String)  
Print\_Pages\_Down (PagesDown:Numeric)  
Print\_Pages\_Across (PagesAcross:Numeric)  
Print\_Orientation (Setting:String)  
Print\_Page\_Breaks (Yes?:Enumeration {Yes!; No!})  
Print\_PageSetupReset ()  
Print\_Paper\_Type (PaperSize:String)  
Print\_Print\_To\_Fit (Yes?:Enumeration {Yes!; No!})  
Print\_Right\_Margin (Margin:String)  
Print\_Scaling (PercentageValue:Numeric)  
Print\_Top\_Margin (Margin:String)

### Description

These command options affect the 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.

{Print.Options_Dialog}	Displays the Page Setup dialog.
{Print.Bottom_Margin Value}	Sets the amount of space between the edge of the page and the bottom of the document
{Print.CreateFooter Yes  No}	Determines whether your print selection contains a footer.
{Print.CreateHeader Yes  No}	Determines whether your print block contains a header.
{Print.Footer <i>FooterString</i> }	Creates and specifies text for a footer
{Print.Footer_Margin Value}	Sets the amount of space between the last row of data and the footer
{Print.Footers_Font "Typeface, PointSize, Bold(Yes No), Italic(Yes No), Underline(Yes No), Strikeout(Yes No)"}	Specifies the typeface, point size, and type style for footer text
{Print.Header <i>HeaderString</i> }	Creates and specifies text for a header
{Print.Header_Margin Value}	Sets the amount of space between the header and the first row of data
{Print.Headers_Font "Typeface, PointSize, Bold (Yes No), Italic (Yes No), Underline (Yes No), Strikeout (Yes No)"}	Specifies the typeface, point size, and type style for header text
{Print.Left_Margin Value}	Specifies the amount of space between the edge of the page and the left of the document
{Print.PagesDown N}	Determines how many pages long a print selection will occupy.

{Print.PagesAcross N}	Determines how many pages wide a print selection will occupy.
{Print.Orientation Landscape Portrait}	Specifies portrait or landscape printing orientation
{Print.Page_Breaks Yes No}	Starts a new printed page at each soft page break
{Print.PageSetupReset}	Resets the dialog box to its default settings, replacing all selections in the dialog box
{Print.Paper_Type PaperSize}	Controls the paper type and printing orientation
{Print.Print_To_Fit Yes No}	Specifies the maximum width and height in pages to use when printing the print selection
{Print.Right_Margin Value}	Specifies the amount of space between the edge of the page and the right of the document
{Print.Scaling 1-1000}	Specifies the percentage to increase or decrease the size of notebook data on the printed page
{Print.Top_Margin Value}	Specifies the amount of space between the edge of the page and the top of the document

### Example

This 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"}
```

#### ► Related topics



## Print Command Options

### PerfectScript Syntax

Print\_All\_Pages (Yes?:Enumeration {Yes!; No!})  
Print\_Area (Area:String)  
Print\_Block (Block:String)  
Print\_Copies (Number:Numeric)  
Print\_DoPrint ()  
Print\_DoPrintGraph ()  
Print\_End\_Page\_Number (PageNumber:Numeric)  
Print\_Group\_Copies (Group:String)  
Print\_Start\_Page\_Number (PageNumber:Numeric)  
PrinterSetup (Printer:String; Port:String; PrintToFile?:Enumeration {Yes!; No!}; Filename:String;  
ReplaceOption:Enumeration {Cancel!; Overwrite!; Backup!; Append!})

### Description

These command options affect printing. {Print.DoPrint} prints the active notebook (or active chart) using current print settings. {Print.DoPrintGraph} provides a quick way to print a chart. If a floating chart is selected, {Print.DoPrintGraph} prints the chart being shown; if a chart icon is selected, {Print.DoPrintGraph} prints the chart represented by that icon; if a chart window is active, {Print.DoPrintGraph} prints the chart shown.

{Print.All_Pages Yes No}	Prints all notebook pages
{Print.Area Notebook   Selection   Current Sheet}	Specifies how much of a notebook to print
{Print.Block <i>Block</i> }	Prints the cells you specify
{Print.Copies <i>Value</i> }	Specifies the number of copies to print
{Print.DoPrint}	Sends the document to the printer
{Print.DoPrintGraph}	Prints the selected chart
{Print.GroupCopies 0 1}	Prints multiple copies sorted by sets of copies. Will "collate" copies when set to zero, and "group" copies when set to 1.
{Print.Start_Page_Number <i>Value</i> }	Specifies the beginning and ending pages in the document to print
{Print.PrinterSetup <i>Printer; Port; PrintToFile</i> (0 1); <i>Filename; CancelOverwrite</i> (0)   <i>Replace</i> (1)   <i>Backup</i> (2)   <i>Append</i> (3)}	Lets you specify details of the printing process

### Example

This macro selects an icon on the Objects sheet named Report3 and prints the chart it represents.

```
{OBJECTSPAGEGOTO}  
{SELECTOBJECT Report3}  
{Print.DoPrintGraph}
```

This macro prints pages 7 through 12 of a document. The print selection is A3..C234.

```
{Print.Block A3..C234}  
{Print.All_Pages No}  
{Print.Start_Page_Number 7}  
{Print.End_Page_Number 12}  
{Print.DoPrint}
```

## Page Formatting Command Options

### PerfectScript Syntax

Print\_Between\_Block\_Formatting (Space:String)  
Print\_Between\_Page\_Formatting (Space:String)  
Print\_Cell\_Formulas (Yes?:Enumeration {Yes!; No!})  
Print\_Center\_Block (Yes?:Enumeration {Yes!; No!})  
Print\_Left\_Heading (Block:String)  
Print\_Lines\_Between\_Blocks (Lines:Numeric)  
Print\_Lines\_Between\_Pages (Lines:Numeric)  
Print\_Print\_Borders (Yes?:Enumeration {Yes!; No!})  
Print\_Print\_Gridlines (Yes?:Enumeration {Yes!; No!})  
Print\_PrinterSetup (Printer:String; Port:String; PrintToFile?:String; Filename:String; OverWrite?:String)  
Print\_PrintOptionsReset ()  
Print\_PrintReset ()  
Print\_Top\_Heading (Block:String)

### Description

These command options affect spreadsheet printing. {Print.Between\_Page\_Formatting} and {Print.Lines\_Between\_Pages} control the amount of space left between notebook sheets (if the print selection spans multiple sheets).

{Print.Between\_Block\_Formatting} and {Print.Lines\_Between\_Blocks} control space between the selections that make up a noncontiguous print selection.

{Print.Between_Block_Formatting "Lines" "Page Advance"}	Separates groups of cells with blank lines or page breaks
{Print.Between_Page_Formatting "Lines" "Page Advance"}	Separates sheets of 3-D cells with blank lines or page breaks
{Print.Cell_Formulas Yes No}	Prints each cell's address and contents instead of its calculated results
{Print.Center_Block Yes No}	Centers the cells of the print selection between the left and right margins of the printed page
{Print.Left_Heading <i>Block</i> }	Adds the cell entries you specify as headings to print at the left of each printed page
{Print.Lines_Between_Blocks <i>Value</i> }	Specifies how many blank lines to print between each group of cells
{Print.Lines_Between_Pages <i>Value</i> }	Specifies how many blank lines to print between each sheet of 3-D pages
{Print.Print_Borders Yes No}	Includes row and column borders in the printed document
{Print.Print_Gridlines Yes No}	Includes the spreadsheet grid in the printed document
{Print.PrintOptionsReset}	Resets the dialog box to its default settings, replacing all selections in the dialog box
{Print.Top_Heading <i>Block</i> }	Adds the cell entries you specify as headings to print at the top of each printed page
{Print.PrintReset}	Resets all print settings

### Example

This macro specifies that three lines should be printed between each notebook sheet (if the print selection spans multiple sheets), and that row and column borders should print.

```
{Print.Between_Page_Formatting "Lines"}  
{Print.Lines_Between_Pages 3}  
{Print.Print_Borders Yes}
```

### ► Related topics

## {PTTESTM}

### Syntax

{PTTESTM *InBlock1*, *InBlock2*, *OutBlock*, <*Labels*>, <*Alpha*>, <*Difference*>}

### PerfectScript Syntax

PTTESTM (InBlock1:String; InBlock2:String; OutBlock:String; [Labels?:Enumeration {Yes!, No!}]; [Alpha:Numeric]; [Difference:Numeric])

### Description

{PTTESTM} performs a paired two-sample Student's t-Test for means. Each value from *InBlock1* is paired with a value from *InBlock2*. *InBlock1* and *InBlock2* must have the same number of values.

{PTTESTM} is equivalent to the t-Test analysis tool.

### Parameters

<i>InBlock1</i>	The first input cells containing a column or row of numeric values
<i>InBlock2</i>	The second input cells containing a column or row of numeric values
<i>OutBlock</i>	Upper-left cell of the output cells
<i>Labels</i>	1 if labels are located in the first column or row of the input cells; 0 if the input cells do not contain labels; the default is 0
<i>Alpha</i>	Significance level of the test; the default is 0.05
<i>Difference</i>	Hypothetical mean difference; the default is 0

### ► Related topics

## **{PTTESTV}**

### **Syntax**

{PTTESTV *InBlock1*,*InBlock2*,*OutBlock*,<*Labels*>,<*Alpha*>}

### **PerfectScript Syntax**

PTTESTV (*InBlock1*:String; *InBlock2*:String; *OutBlock*:String; [*Labels?*:Enumeration {Yes!; No!}]; [*Alpha*:Numeric])

### **Description**

{PTTESTV} performs a Student's t-Test using two independent (rather than paired) samples with unequal variances. {PTTESTV} is equivalent to the t-Test analysis tool.

### **Parameters**

<i>InBlock1</i>	The first input cells containing a column or row of numeric values
<i>InBlock2</i>	The second input cells containing a column or row of numeric values
<i>OutBlock</i>	Upper-left cell of the output cells
<i>Labels</i>	1 if labels are located in the first column or row of the input cells; 0 if the input cells do not contain labels; the default is 0
<i>Alpha</i>	Significance level of the test; the default is 0.05

### **▶ Related topics**

# {PUT}

## Syntax

{PUT *Location*,*Column#*,*Row#*,*Value*<:*Type*> }

## PerfectScript Syntax

Put (Block:String; Column:Numeric; Row:Numeric; Value:Any)

## Description

{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 is 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#* exceeds the number of columns or rows in the cells, the macro stops. [{ONERROR}](#) cannot trap this error.)

## Example

Each of the following examples assumes cell A41 contains the value 25, the selection 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 cell 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 cell numbers (cell B49).

{PUT numbers,1,6,data} copies the contents of the cell data to the 2nd column and 7th row of numbers (cell B50). If there is no selection named data, this example instead places a label ("data") into cell B50.

## Parameters

<i>Location</i>	Cells within which Value will be stored, either as a value or label, as specified by Type
<i>Column#</i>	Number of columns into the specified cells to store Value
<i>Row#</i>	Number of rows into the specified cells to store Value
<i>Value</i>	String or numeric value
<i>Type</i>	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)

## ► [Related topics](#)

## {PUTBLOCK}

### Syntax

{PUTBLOCK *Data*,<*Block*>,<*Date?*(0|1)>}

### PerfectScript Syntax

PutBlock (Data:Any; [Block:String]; [Date?:Enumeration {Yes!; No!}])

### Description

{PUTBLOCK} lets you quickly enter the same value, label, or formula in multiple cells. *Data* is a string or value to place in *Block*. If *Block* is not specified, the currently selected cells are used. *Block* can be noncontiguous; if so, be sure to enclose it in parentheses. If *Data* is a formula containing relative addresses, those addresses are adjusted automatically.

### Example

{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.

{PUTBLOCK "11/01/94", (A:D3,B:D3,C:D3,D:D3),1} enters the date 11/01/94 in cell D3 of sheets A through D.

### Parameters

<i>Data</i>	Entry to type
<i>Block</i>	Cells to type <i>Data</i> in (optional)
<i>Date?</i>	Whether to enter <i>Data</i> as a date (1) or a label (0)

### ▶ Related topics

## {PUTBLOCK2}

### Syntax

{PUTBLOCK2 *Data*,<*Block*>}

### PerfectScript Syntax

PutBlock2 (Data:Any; [Block:String])

### Description

{PUTBLOCK2} enters the same value, label, or formula in multiple cells like {PUTBLOCK} but parses date formats automatically and requires a formula prefix before numeric values. *Data* is a string or value to place in *Block*. If *Block* is not specified, the currently selected cells are used. *Block* can be noncontiguous; if so, be sure to enclose it in parentheses. If *Data* is a formula containing relative addresses, those addresses are adjusted automatically.

### Example

{PUTBLOCK2 "Quarter 1",A..D:A1} enters the label Quarter 1 in cells A:A1 through D:A1.

{PUTBLOCK2 +1990,A..D:B1} enters the value 1990 in cells A:B1 through D:B1.

{PUTBLOCK2 "+A1",C3..C12) enters the formula +A1 in C3, +A2 in C4, and so on.

{PUTBLOCK2 "11/01/94", (A:D3,B:D3,C:D3,D:D3)} enters the date 11/01/94 in cell D3 of sheets A through D.

### Parameters

<i>Data</i>	Entry to type
<i>Block</i>	Cells to type <i>Data</i> in (optional)

### ► Related topics

## **{PUTCELL}**

### **Syntax**

{PUTCELL *Data*,<*Date?*(0|1)>}

### **PerfectScript Syntax**

PutCell (*Data*:Any; [*Date?*:Enumeration {Yes!; No!}])

### **Description**

{PUTCELL} is an easy way to store information in the active cell.

### **Example**

{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.

{PUTCELL "11/01/94",1} stores the date 11/01/94 in the active cell

### **Parameters**

<i>Data</i>	String to type into the active cell
<i>Date?</i>	Whether to enter <i>Data</i> as a date (1) or a label (0)

### **▶ Related topics**



## **{PUTCELL2}**

### **Syntax**

{PUTCELL2 *Data*}

### **PerfectScript Syntax**

PutCell2 (Data:Any)

### **Description**

{PUTCELL2} stores information in the active cell like {PUTCELL} but parses date formats automatically and requires a formula prefix before numeric values.

### **Example**

{PUTCELL2 "Peggy Danderhoff"} stores Peggy Danderhoff as a label in the active cell.

{PUTCELL2 +45067} stores the number 45067 as a value in the active cell.

{PUTCELL2 "@SUM(A1..A27)"} stores the formula @SUM(A1..A27) in the active cell.

{PUTCELL2 "11/01/94"} stores the date 11/01/94 in the active cell

### **Parameters**

*Data*      String to type into the active cell

### **► Related topics**



## **{QGOTO}**

### **Description**

{QGOTO} displays and selects the specified cells. This command is equivalent to the Go To key, F5. A parameter, ~CellName, must be used after {QGOTO} to set the specified cell in the Go To dialog box. You can also use the command equivalent {EditGoto *Block*}.

A related command, {GOTO}, moves to the upper-left cell of the destination cells, but does not select the cells.

### **Example**

{QGOTO}~D12 will select cell D12.

### **▶ Related topics**

## **{QUERY}**

### **Description**

{QUERY} repeats the last Notebook Query operation performed.

▶ **Related topics**

## {Query}

### Syntax

{Query.Option}

### PerfectScript Syntax

Query\_Assign\_Names ()  
Query\_Criteria\_Table (Block:String)  
Query\_Database\_Block (Block:String)  
Query\_Delete ()  
Query\_EndLocate ()  
Query\_Extract ()  
Query\_Locate ()  
Query\_Output\_Block (Block:String)  
Query\_Reset ()  
Query\_Unique ()

### Description

{Query} lets you set up a Quattro Pro database and search for records in that database. {Query.Locate} enters FIND mode and stays under macro control until {PAUSEMACRO} is used or {Query.EndLocate}, which exits FIND mode.

You can use {Query?} or {Query!} to display the Notebook Data Query dialog box. {Query?} lets you manipulate the dialog box, whereas {Query!} relies on the macro to manipulate it.

### Example

The following macro sets up database cells and criteria table (A2..G37 and H1..H2), searches for records using the criteria table, sets up output cells at J2..P2, and copies any records found there.

```
{Query.Database_Block A2..G37}  
{Query.Criteria_Table H1..H2}  
{Query.Locate}  
{Query.EndLocate}  
{Query.Output_Block J2..P2}  
{Query.Extract}
```

### Options

{Query.Assign_Names}	Assigns cell names to fields so you can use them in search queries
{Query.Criteria_Table <i>Block</i> }	Specifies cells containing search conditions, including field names
{Query.Database_Block <i>Block</i> }	Specifies the data, including field names, to search
{Query.Delete}	Deletes all records that meet the search criteria
{Query.Extract}	Copies all records that meet the search criteria to the output cells
{Query.Locate}	Highlights all records that meet the search criteria
{Query.Output_Block <i>Block</i> }	Specifies the cells where you want to copy records and field names that meet the search criteria
{Query.Reset}	Removes all selection settings
{Query.Unique}	Copies records like Extract, but skips duplicate records

## {QuickCorrect}

### Syntax

{QuickCorrect I|0}

## **PerfectScript Syntax**

QuickCorrect (Enable?:Enumeration {1!; 0!})

### **Description**

{QuickCorrect} replaces common spelling errors and mistyped words; it can also be used to automatically expand abbreviations. {QuickCorrect 1} activates the QuickCorrect feature; {QuickCorrect 0} turns it off.

## {QuickFilter.Go}

### Syntax

{QuickFilter.Go *CellReference*; <*OpCode1*; *Arg1*; *Conditional1*; *OpCode2*; *Arg2*; *Conditional2*; *OpCode3*; *Arg3*>}

### PerfectScript Syntax

QuickFilter\_Go ([Block?:String]; [OpCode1?:String]; [Value1?:String]; [Conditional1?:String]; [OpCode2?:String]; [Value2?:String]; [Conditional2?:String]; [OpCode3?:String]; [Value3?:String])

### Description

Performs QuickFilter operations on cells. You can have 2, 5, or 7 optional args.

### Example:

```
{QuickFilter.Go A:A1}
```

Equivalent to "Show All." Flushes ALL filters associated with Column A

```
{QuickFilter.Go A:B5;equal to""}
```

Equivalent to "Blanks." Filters all rows out except for those with blanks in Column B.

```
{QuickFilter.Go A:F24;not equal to""}
```

Equivalent to "Non Blanks." Filters out all rows except for those without blanks in Column F.

### Parameters

<i>OpCode#</i>	"Equal to," or "not equal to," "greater than," "less than," "greater than or equal to," "less than or equal to," "begins with," "does not begin with," "ends with," "does not end with," "contains," "does not contain."
<i>Arg#</i>	Can be numeric, or a string. Wildcards are not valid.
<i>Conditiona l#</i>	AND or OR

### ► Related topics

## **{QuickFilter.Toggle}**

### **Syntax**

{QuickFilter.Toggle <Block>}

### **PerfectScript Syntax**

QuickFilter\_Toggle ([Block?:String])

### **Description**

Turns on/off QuickFilters for the current cells.

### **▶ Related topics**



## {QuickFilter.TopGo}

### Syntax

```
{QuickFilter.TopGo CellReference; <OpCode; Arg>}
```

### PerfectScript Syntax

```
QuickFilter_TopGo ([Block?:String]; [OpCode1?:String]; [Value1?:Numeric])
```

### Description

Performs QuickFilter operations on cells.

### Example:

```
{QuickFilter.TopGo A:C51;top value;10}
```

Equivalent to Top Ten Values. Filters out all rows except for those that contain the top 10 values in column C.

```
{QuickFilter.TopGo A:E17;bottom percent;23}
```

Equivalent to Bottom 23 Percent. Filters out all rows except for those that contain the bottom 23% in column E.

### Parameters

<i>OpCode</i>	"Top value," "top percent," "bottom value," "bottom percent"
<i>Arg</i>	Must be numeric. Wildcards are not valid.

### ► Related topics

## {QuickFunction}

### Syntax

{QuickFunction Name; <Block>}

### PerfectScript Syntax

QuickFucntion(Name?: String!, Block?: <Block>)

### Description

{QuickFunction} is equivalent to selecting cells and clicking the QuickFunction button on the toolbar. Block includes rows and/or columns to sum plus adjacent empty cells to hold the results. The default Block is the current selection.

### Parameters

<i>Name</i>	SUM, MIN, MAX, AVG, PUREAVG, MULT, PMT, RATE, IRATE, TERM, PV, FV
<i>Block</i>	A database block including field labels and records

### ► Related topics

## **{QUIT}**

### **Description**

{QUIT} ends all macro execution, and returns control of Quattro Pro to you.

Use {QUIT} in conjunction with {IF}, {LOOK}, {MENUBRANCH}, or {MENUCALL} to end a macro under user control.

### **Example**

The following macro displays a menu that has a "Quit" option, which returns you to Ready mode.

```
quit_menu      Continue          Quit
Keep going     Quit to Ready mode
{BRANCH \G}    {QUIT}
\G{MENUBRANCH quit_menu}
```

### **► Related topics**



# {RANDOM}

## Description

{RANDOM} generates cells of random values drawn from a selected distribution. It is equivalent to the Random Number analysis tool. {RANDOM} has a different format for the following distribution types:

Uniform

Every value has an equal probability of being selected.

Normal

Has the qualities of a symmetrical, bell-shaped curve.

Bernoulli

Has two possible outcomes, failure or success, represented by 0 and 1.

Binomial

Represents the distribution of successful outcomes in a given number of independent Bernoulli trials.

Poisson

The distribution of values in any interval depends on the length of the interval and the constant Lambda, the expected number of occurrences in an interval

Patterned

A pattern of repeated values and sequences.

Discrete

Every value in designated cells has a specified probability of being selected (the cumulative probabilities equal 1).

## ► Related topics

## {RANDOM} - Uniform Distribution

### Syntax

{RANDOM *OutBlock*, *Columns*, *Rows*, 1, *Seed*, *LowerBound*, *UpperBound*}

### PerfectScript Syntax

RANDOM (OutBlock:String; Columns:Numeric; Rows:Numeric; Type:Enumeration {Uniform!; Normal!; Bernoulli!; Binomial!; Poisson!; Patterned!; Discrete!}; Seed:Numeric; Parameter1:Any; [Parameter2:Numeric]; [Parameter3:Numeric]; [Parameter4:Numeric]; [Parameter5:Numeric])

### Description

When the *Distribution* argument equals 1, {RANDOM} generates random values drawn from a uniform distribution.

### Parameters

<i>OutBlock</i>	Upper-left cell of the output cells
<i>Columns</i>	A value indicating the number of random-number sets to generate; default is the number of columns in <i>OutBlock</i>
<i>Rows</i>	A value indicating the number of rows of random numbers to generate for each column
1	Indicates uniform distribution
<i>Seed</i>	Starting number for the random-number-generation algorithm
<i>LowerBound</i>	A value indicating the lower bound on the set of numbers to generate
<i>d</i>	
<i>UpperBound</i>	A value indicating the upper bound on the set of numbers to generate
<i>d</i>	

### ► Related topics

## {RANDOM} - Normal Distribution

### Syntax

{RANDOM *OutBlock*, *Columns*, *Rows*, 2, *Seed*, *Mean*, *SDev*}

### PerfectScript Syntax

RANDOM (OutBlock:String; Columns:Numeric; Rows:Numeric; Type:Enumeration {Uniform!; Normal!; Bernoulli!; Binomial!; Poisson!; Patterned!; Discrete!}); Seed:Numeric; Parameter1:Any; [Parameter2:Numeric]; [Parameter3:Numeric]; [Parameter4:Numeric]; [Parameter5:Numeric])

### Description

When the *Distribution* argument equals 2, {RANDOM} generates random values drawn from a normal distribution.

### Parameters

<i>OutBlock</i>	Upper-left cell of the output cells
<i>Columns</i>	A value indicating the number of random-number sets to generate; default is the number of columns in <i>OutBlock</i>
<i>Rows</i>	A value indicating the number of rows of random numbers to generate for each column
2	Indicates normal distribution
<i>Seed</i>	Starting number for the random-number-generation algorithm
<i>Mean</i>	A value indicating the mean of the set of numbers to generate
<i>SDev</i>	A value indicating the standard deviation of the set of numbers to generate

### ► Related topics

## {RANDOM} - Bernoulli Distribution

### Syntax

{RANDOM *OutBlock*, *Columns*, *Rows*, 3, *Seed*, *Prob*}

### PerfectScript Syntax

RANDOM (OutBlock:String; Columns:Numeric; Rows:Numeric; Type:Enumeration {Uniform!; Normal!; Bernoulli!; Binomial!; Poisson!; Patterned!; Discrete!}; Seed:Numeric; Parameter1:Any; [Parameter2:Numeric]; [Parameter3:Numeric]; [Parameter4:Numeric]; [Parameter5:Numeric])

### Description

When the *Distribution* argument equals 3, {RANDOM} generates random values drawn from a Bernoulli distribution.

### Parameters

<i>OutBlock</i>	Upper-left cell of the output cells
<i>Columns</i>	A value indicating the number of random-number sets to generate; default is the number of columns in <i>OutBlock</i>
<i>Rows</i>	A value indicating the number of rows of random numbers to generate for each column
3	Indicates Bernoulli distribution
<i>Prob</i>	Starting number for the random-number-generation algorithm
<i>Seed</i>	A value indicating the probability of success on each trial run; must be greater than or equal to 0 and less than or equal to 1

### ► Related topics



## {RANDOM} - Binomial Distribution

### Syntax

{RANDOM *OutBlock*, *Columns*, *Rows*, 4, *Seed*, *Prob*, *Trials*}

### PerfectScript Syntax

RANDOM (OutBlock:String; Columns:Numeric; Rows:Numeric; Type:Enumeration {Uniform!; Normal!; Bernoulli!; Binomial!; Poisson!; Patterned!; Discrete!}; Seed:Numeric; Parameter1:Any; [Parameter2:Numeric]; [Parameter3:Numeric]; [Parameter4:Numeric]; [Parameter5:Numeric])

### Description

When the *Distribution* argument equals 4, {RANDOM} generates random values drawn from a binomial distribution.

### Parameters

<i>OutBlock</i>	Upper-left cell of the output cells
<i>Columns</i>	A value indicating the number of random-number sets to generate; default is the number of columns in <i>OutBlock</i>
<i>Rows</i>	A value indicating the number of rows of random numbers to generate for each column
4	Indicates binomial distribution
<i>Seed</i>	Starting number for the random-number-generation algorithm
<i>Prob</i>	A value indicating the probability of success on each trial run; must be greater than or equal to 0 and less than or equal to 1
<i>Trials</i>	A value indicating the number of trials

### ► Related topics

## {RANDOM} - Poisson Distribution

### Syntax

{RANDOM *OutBlock*, *Columns*, *Rows*, 5, *Seed*, *Lambda*}

### PerfectScript Syntax

RANDOM (OutBlock:String; Columns:Numeric; Rows:Numeric; Type:Enumeration {Uniform!; Normal!; Bernoulli!; Binomial!; Poisson!; Patterned!; Discrete!}; Seed:Numeric; Parameter1:Any; [Parameter2:Numeric]; [Parameter3:Numeric]; [Parameter4:Numeric]; [Parameter5:Numeric])

### Description

When the *Distribution* argument equals 5, {RANDOM} generates random values drawn from a Poisson distribution.

### Parameters

<i>OutBlock</i>	Upper-left cell of the output cells
<i>Columns</i>	A value indicating the number of random-number sets to generate; default is the number of columns in <i>OutBlock</i>
<i>Rows</i>	A value indicating the number of rows of random numbers to generate for each column
5	Indicates Poisson distribution
<i>Seed</i>	Starting number for the random-number-generation algorithm
<i>Lambda</i>	A parameter to the Poisson distribution representing the expected number of events in each unit

### ► Related topics

## {RANDOM} - Patterned Distribution

### Syntax

{RANDOM *OutBlock*, *Columns*, *Rows*, 6, *Seed*, *LowerBound*, *UpperBound*, *Step*, *RepeatNumber*, *RepeatSequence*}

### PerfectScript Syntax

RANDOM (OutBlock:String; Columns:Numeric; Rows:Numeric; Type:Enumeration {Uniform!; Normal!; Bernoulli!; Binomial!; Poisson!; Patterned!; Discrete!}; Seed:Numeric; Parameter1:Any; [Parameter2:Numeric]; [Parameter3:Numeric]; [Parameter4:Numeric]; [Parameter5:Numeric])

### Description

When the *Distribution* argument equals 6, {RANDOM} generates random values drawn from a patterned distribution.

### Parameters

<i>OutBlock</i>	Upper-left cell of the output cells
<i>Columns</i>	A value indicating the number of random-number sets to generate; default is the number of columns in <i>OutBlock</i>
<i>Rows</i>	A value indicating the number of rows of random numbers to generate for each column
6	Indicates patterned distribution
<i>Seed</i>	Starting number for the random-number-generation algorithm
<i>LowerBound</i>	A value indicating the lower bound on the set of numbers to generate
<i>UpperBound</i>	A value indicating the upper bound on the set of numbers to generate
<i>Step</i>	Increment value between <i>LowerBound</i> and <i>UpperBound</i>
<i>RepeatNumber</i>	A value indicating the number of times to repeat each value
<i>RepeatSequence</i>	A value indicating the number of times to repeat each sequence of values

### ► Related topics

## {RANDOM} - Discrete Distribution

### Syntax

{RANDOM *OutBlock*, *Columns*, *Rows*, 7, *Seed*, *InBlock*}

### PerfectScript Syntax

RANDOM (OutBlock:String; Columns:Numeric; Rows:Numeric; Type:Enumeration {Uniform!; Normal!; Bernoulli!; Binomial!; Poisson!; Patterned!; Discrete!}); Seed:Numeric; Parameter1:Any; [Parameter2:Numeric]; [Parameter3:Numeric]; [Parameter4:Numeric]; [Parameter5:Numeric])

### Description

When the *Distribution* argument equals 7, {RANDOM} generates random values drawn from a discrete distribution.

### Parameters

<i>OutBlock</i>	Upper-left cell of the output cells
<i>Columns</i>	A value indicating the number of random-number sets to generate; default is the number of columns in <i>OutBlock</i>
<i>Rows</i>	A value indicating the number of rows of random numbers to generate for each column
7	Indicates discrete distribution
<i>Seed</i>	Starting number for the random-number-generation algorithm
<i>InBlock</i>	One or more numeric cell values representing the input cells, which contain a range of values and their probabilities, each in a separate column

### ► Related topics

## **{RANKPERC}**

### **Syntax**

{RANKPERC *InBlock*,*OutBlock*,<*Grouped*>,<*Labels*(0|1)>}

### **PerfectScript Syntax**

RANKPERC (InBlock:String; OutBlock:String; [Grouped:String]; [Labels?:Enumeration {Yes!; No!}])

### **Description**

{RANKPERC} returns the ordinal and percent rank of each value in *InBlock*. {RANKPERC} is equivalent to the Rank and Percentile analysis tool.

### **Parameters**

<i>InBlock</i>	Input cells containing one or more columns or rows of numeric values
<i>OutBlock</i>	Upper-left cell of the output cells
<i>Grouped</i>	"C" to group results by column or "R" to group results by row; the default is "C"
<i>Labels</i>	1 if labels are located in the first column or row of the input cells; 0 if the input cells do not contain labels; the default is 0

### **► Related topics**

## **{READ}**

### **Syntax**

{READ #Bytes,Location}

### **Description**

{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. (See [{GETPOS}](#) for a discussion of the file pointer.)

{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 will 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.

### **Example**

This macro 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 28
name_length14
phone_length 12
name Hall, Sue Ann
phone617-555-5678
```

### **Parameters**

#Bytes	Number of bytes of characters to read from a file
Location	Cell in which to store the characters read

### **▶ [Related topics](#)**

## **{READLN}**

### **Syntax**

{READLN *Location*}

### **Description**

{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 does not read the carriage return/linefeed into the cell. (See [{GETPOS}](#) for a discussion of the file pointer.)

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 is 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.

### **Example**

The following macro opens the text file TEST.TXT, reads in a line, and calculates the length of the line by subtracting the starting position from the ending position, 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}
```

```
start0
end 22
num_char 21
inputThis is a short line.
```

### **Parameters**

*Location*            Cell in which to store the characters read

#### **▶ Related topics**

# {RECALC}

## Syntax

{RECALC *Location*,<*Condition*>,<*Iteration#*>}

## PerfectScript Syntax

Recalc (Block:String; [Condition:Any]; [Iteration:Numeric])

## Description

{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 cells repeatedly until the specified condition is met. You can also supply *Iteration#* to specify the maximum number of times to recalculate formulas trying to satisfy *Condition*. To use *Iteration#*, *Condition* 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 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.

If there are formulas within the cells being recalculated that depend on formulas outside of the cells, they might not evaluate correctly. Make sure *Location* encompasses all the cells referenced by formulas within the cells.

## Parameters

<i>Location</i>	Cells to recalculate
<i>Condition</i>	Condition to be met before recalculation is halted (optional)
<i>Iteration</i>	Maximum number of times to recalculate <i>Location</i> trying to meet
<i>#</i>	<i>Condition</i> (optional)

## ► [Related topics](#)



## {RECALCCOL}

### Syntax

{RECALCCOL *Location*,<*Condition*>,<*Iteration#*>}

### PerfectScript Syntax

RecalcCol (Block:String; [Condition:Any]; [Iteration:Numeric])

### Description

{RECALCCOL} recalculates the specified portion of a notebook in column-by-column order. It is similar to [{RECALC}](#), which recalculates row by row. See {RECALC} for information on when {RECALCCOL} is appropriate and when you need to use [{CALC}](#) instead.

### Parameters

<i>Location</i>	Cells to recalculate
<i>Condition</i>	Condition to be met before recalculation is halted (optional)
<i>Iteration</i>	Maximum number of times to recalculate <i>Location</i> trying to meet
<i>#</i>	<i>Condition</i> (optional)

### ► [Related topics](#)

## **{RefreshMenuBar}**

### **Syntax**

{RefreshMenuBar}

### **PerfectScript Syntax**

RefreshMenuBar ()

### **Description**

Refreshes the menu bar.

## **{RefreshScreenOn} and {RefreshScreenOff}**

### **Description**

{RefreshScreenOn} and {RefreshScreenOff} turn on or off the repainting of Quattro Pro behind an active dialog box.

# {REGRESS}

## Syntax

{REGRESS *InBlockY*, *InBlockX*, *YIntZero*(0|1), *Labels*(0|1), *Confidence*, *SumOutBlock*, *Residuals*(0|1), *StdResiduals*(0|1), <*ResidualOutBlock*>, <*ProbOutBlock*>}

## PerfectScript Syntax

REGRESS (*InBlockY*:String; *InBlockX*:String; *YIntZero*?:Enumeration {Yes!; No!}; *Labels*?:Enumeration {Yes!; No!}; *Confidence*:Numeric; *SumOutBlock*:String; *Residuals*?:Enumeration {Yes!; No!}; *StdResiduals*?:Enumeration {Yes!; No!}; [*ResidualOutBlock*:String]; [*ProbOutBlock*:String])

## Description

{REGRESS} performs multiple linear regression analysis. {REGRESS} is equivalent to the Advanced Regression analysis tool.

## Parameters

<i>InBlockY</i>	Input cells containing a single column of y values (the dependent variables)
<i>InBlockX</i>	Input cells containing one or more columns of x values (the independent variables)
<i>YIntZero</i>	1 if the y-intercept is 0 (the line of regression passes through the origin); 0 if the y-intercept is not 0
<i>Labels</i>	1 if labels are located in the first column or row of the <i>InBlockY</i> and <i>InBlockX</i> ; 0 if the input selections do not contain labels
<i>Confidence</i>	A value indicating the confidence level to apply to the regression
<i>SumOutBlock</i>	Upper-left cell of the output cells for the summary table (allow at least seven columns)
<i>Residuals</i>	1 or 0; if 1, includes residuals in the output table
<i>StdResiduals</i>	1 or 0; if 1, includes standardized residuals in the output table
<i>ResidualOutBlock</i>	Upper-left cell of the output cells for the residuals table (allow at least four columns)
<i>ProbOutBlock</i>	Upper-left cell of the output cells for the probabilities table (allow at least two columns)

## ► Related topics

## {Regression}

### Syntax

{Regression.Option}

### PerfectScript Syntax

Regression\_Dependent (Block:String)

Regression\_Go ()

Regression\_Independent (Block:String)

Regression\_Output (Block:String)

Regression\_Reset ()

Regression\_Y\_Intercept (Mode:String)

### Description

{Regression} performs a regression analysis to show the relationship between a set of independent variables and a dependent variable.

{Regression.Dependent} indicates the dependent-variable cells. {Regression.Independent} defines the independent variables. In {Regression.Independent}, *Block* can be noncontiguous with one variable to a column. The dependent and independent selections 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 selections, use {Regression.Go} again to calculate a new regression table.

You can use {Regression?} or {Regression!} to display the Linear Regression dialog box. {Regression?} lets you manipulate the dialog box, whereas {Regression!} relies on the macro to manipulate it.

### Example

The following macro sets these data selections: Independent, B2..D16; Dependent, F2..F16. The last statement performs the regression analysis and stores the results in the cells with upper-left cell H2.

```
{Regression.Independent A:B2..A:D16}
```

```
{Regression.Dependent A:F2..A:F16}
```

```
{Regression.Output A:H2}
```

```
{Regression.Go}
```

### Options

{Regression.Dependent <i>Block</i> }	Specifies the cells (partial column) containing independent variable (y-axis) data
{Regression.Go}	Performs the regression analysis
{Regression.Independent <i>Block</i> }	Specifies cells containing up to 150 columns of independent variable (x-axis) data
{Regression.Output <i>Block</i> }	Specifies the cells where results will be written
{Regression.Reset}	Clears all regression settings
{Regression.Y_Intercept Compute Zero}	Specifies whether to force the y-intercept value to zero or whether to compute it

## {REQUEST}

### Syntax

{REQUEST DDEChannel,DataToReceive,DestBlock}

### PerfectScript Syntax

Request (DDEChannel:Numeric; DataToReceive:String; DestBlock:String)

### Description

{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 cells such as A2..A7 or a property such as "(Application.Display)". If requesting a property, the property must be enclosed in parentheses. You must use `{INITIATE}` to open a channel of communication and obtain the value *DDEChannel* before using `{REQUEST}`. If your conversation is not 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:

#### Arguments for DataToReceive

String	Purpose
"SysItems"	A listing of all strings you can use with <i>DataToReceive</i> . You can use this command first to view other choices offered by <i>AppName</i> .
"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.

#### Example

This macro gets the major and minor version numbers of GroupWise, which is already running.

```
dde_channel      0
get_vernumber    {INITIATE "GroupWise","Command",dde_channel}
                 {REQUEST
                 dde_channel,"GetOfficeData(ID;MajorVersion!)",G1}
                 {REQUEST
                 dde_channel,"GetOfficeData(ID;MinorVersion!)",G2}
                 {TERMINATE dde_channel}
```

This macro gets information from the fields Task and Completed in ObjectVision file 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}
```

#### Parameters

<i>DDEChannel</i>	DDE channel number of the application to receive data from
<i>DataToReceive</i>	Information to receive from the application
<i>e</i>	
<i>DestBlock</i>	Cells to store the data received into

#### ► Related topics

## **{RESIZE}**

### **Syntax**

{RESIZE *x,y,NewWidth,NewHeight,<VertFlip?>,<HorizFlip?>*}

### **PerfectScript Syntax**

Resize (x:Numeric; y:Numeric; NewWidth:Numeric; NewHeight:Numeric; [VertFlip?:Enumeration {Yes!; No!}]; [HorizFlip?:Enumeration {Yes!; No!}])

### **Description**

{RESIZE} resizes all selected objects in the active window (dialog or chart window).

### **Parameters**

<i>x</i> and <i>y</i>	XY coordinates of the new upper-left corner, in pixels
<i>NewWidt</i>	The new width, in pixels, of the object or group
<i>h</i>	
<i>NewHeig</i>	The new height, in pixels, of the object or group
<i>ht</i>	
<i>VertFlip?</i>	1 if the object or group is flipped vertically from its previous position
<i>HorizFlip?</i>	1 if the object or group is flipped horizontally from its previous position

### **► Related topics**

## **{ResizeToSame}**

### **Description**

{ResizeToSame} lets you resize selected objects in the dialog window to the same size as the first object selected.



## **{RESTART}**

### **Description**

{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.

### **► [Related topics](#)**

## **{RestrictInput}**

### **Syntax**

{RestrictInput.*Option*}

### **PerfectScript Syntax**

RestrictInput\_Enter (Block:String)

RestrictInput\_Exit ()

### **Description**

{RestrictInput.Enter} enters INPUT mode and stays under macro control until {PAUSEMACRO} is used or {RestrictInput.Exit}, which exits INPUT mode.

{RestrictInput.Option} confines selector movement to specific cells of unprotected cells.

You can use {RestrictInput?} or {RestrictInput!} to display the Restrict Input dialog box. {RestrictInput?} lets you manipulate the dialog box, whereas {RestrictInput!} relies on the macro to manipulate it.

### **Options**

{RestrictInput.Enter <i>Block</i> }	Enters INPUT mode and stays under macro control
{RestrictInput.Exit}	Any operation that ends INPUT mode

## **{RETURN}**

### **Description**

{RETURN} ends the executing subroutine and returns control to the macro that called it. If the macro executing is not 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 with {IF} to return to the main macro if a certain condition is met.

See {Subroutine} for an example of using {RETURN}.

### **► Related topics**

## **{ReturnErrorValue}**

### **Syntax**

{ReturnErrorValue}

### **PerfectScript Syntax**

ReturnErrorValue ()

### **Description**

Reinstates the ability for Quattro Pro to return a specific error value, if one is warranted.

## **{RIGHT} and {R}**

### **Syntax**

{RIGHT <Number>} or {R <Number>}

### **Description**

{RIGHT} and {R} are equivalent to the Right-arrow key. The optional argument *Number* moves the selector the corresponding number of columns to the right. You can also use cell references or cell names as arguments.

### **Example**

{RIGHT}{RIGHT} moves right two columns.

{R 6} moves right six columns.

{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 selection named count.

### **Parameters**

*Number* Any positive integer (optional)

### **► Related topics**

## {ROWCOLSHOW}

### Syntax

{ROWCOLSHOW *Block*,*Show?*,*Row* or *Col*,*FirstPane?*}

### PerfectScript Syntax

RowColShow (Block:String; Show?:Enumeration {Yes!; No!}); Row?:Enumeration {Yes!; No!};  
FirstPane?:Enumeration {Yes!; No!})

### Description

{ROWCOLSHOW} lets you hide or reveal rows and columns (it is equivalent to the cell 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. 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.

### Example

{ROWCOLSHOW A:A..B,1,0,1} reveals columns A and B on sheet A.

{ROWCOLSHOW A:1..7,0,1,1} hides rows 1 through 7 on sheet A.

{ROWCOLSHOW A:1..7,1,1,0} reveals rows 1 through 7 on sheet A. If the window is split, the rows are revealed in the right or bottom pane.

### Parameters

<i>Block</i>	Cells containing rows or columns to hide or show
<i>Show?</i>	1 to reveal rows or columns; 0 to hide rows or columns
<i>Row</i> or <i>Col</i>	1 to reveal or hide a row; 0 to reveal or hide a column
<i>FirstPane?</i>	1 to affect rows or columns in left or top window pane; 0 to affect them in the right or bottom window pane

### ► Related topics

## {ROWHEIGHT}

### Syntax

{ROWHEIGHT *Block,FirstPane?,Set/Reset,Size*}

### PerfectScript Syntax

RowHeight (Block:String; FirstPane?:Enumeration {Yes!; No!}; Reset?:Enumeration {Yes!; No!}; Size:Numeric)

### Description

{ROWHEIGHT} provides two ways to change the height of a row or rows (it is equivalent to the cell property Row Height). The rows to change are specified by *Block*. *FirstPane?* is used when the active window is split into 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.

*Set/Reset* specifies how to change the height. To set a row height, use this syntax: {ROWHEIGHT *Block, FirstPane?, 0, Size*}

*Size* is the new row height, in twips. 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 this syntax: {ROWHEIGHT *Block, FirstPane?, 1*}

### Example

{ROWHEIGHT A:1..A:2,1,0,1440} sets the height of rows 1 and 2 (on sheet A) to one inch (1,440 twips).

{ROWHEIGHT A:1..A:2,0,0,2160} sets the height of rows 1 and 2 (on sheet 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 sheet A) to the default height.

### Parameters

<i>Block</i>	Cells containing rows to resize
<i>FirstPane?</i>	1 to resize rows in left or top window pane; 0 to resize rows in right or bottom window pane
<i>Set/Reset</i>	0 to set the row height; 1 to reset the row height
<i>Size</i>	New height (in twips) if setting size; not needed if resetting size

### ► Related topics



## **{SAMPLE}**

### **Syntax**

{SAMPLE *InBlock,OutBlock,Type,Rate*}

### **PerfectScript Syntax**

SAMPLE (InBlock:String; OutBlock:String; Type:String; Rate:Numeric)

### **Description**

{SAMPLE} returns a periodic or random sample from values in InBlock. {SAMPLE} is equivalent to the Sampling analysis tool.

### **Parameters**

<i>InBlock</i>	One or more numeric or cell values representing the input cells
<i>OutBlock</i>	Upper-left cell of the output cells
<i>Type</i>	"P" to specify periodic sample; "R" to specify random sampling
<i>Rate</i>	A value indicating a sampling rate; if <i>Type</i> = "P", <i>Rate</i> indicates the periodic interval used for sampling; if <i>Type</i> = "R", <i>Rate</i> indicates the number of samples

### **▶ Related topics**



## **{SaveHtml! }**

### **Syntax**

{SaveHtml!}

### **Description**

Lets you open and view the Publish to Internet dialog box.

### **Example**

SaveHtml!

## **{SaveHtml.BackgroundColor}**

### **Syntax**

```
{SaveHtml.BackgroundColor BkColor}
```

### **PerfectScript Syntax**

```
SaveHtml_BackgroundColor (BkColor?: String)
```

### **Description**

Lets you specify the default color of the background.

### **Example**

```
SaveHtml.BackgroundColor Black  
SaveHtml.BackgroundColor "#ff00ff"
```

### **Parameter**

*BkColor*            The name of the default background color

#### **► Note**

- SaveHtml.BackgroundColor will be effective only when the [SaveHtml.UseBrowserColor](#) command is called with 0

## **{SaveHtml.FileOptions}**

### **Syntax**

{SaveHtml.FileOptions *FileData*}

### **PerfectScript Syntax**

SaveHtml\_FileOptions (FileData?: String)

### **Description**

Lets you specify the initial .HTML file name and the default extension to be used. *FileData* consists of two variables delimited by a semicolon.

### **Example**

```
SaveHtml.FileOptions "Index.HTM; HTM"
```

### **Parameters**

<i>FileData</i>	Initial name
[semicolon n delimited ]	The initial .HTML file name  Extension The default extension of .HTML files

## **{SaveHtml.GraphicType}**

### **Syntax**

{SaveHtml.GraphicType *Value*}

### **PerfectScript Syntax**

SaveHtml\_GraphicType (Value?: Numeric)

### **Description**

Lets you specify the file format to use for graphic images.

### **Example**

```
SaveHtml.GraphicType 1
```

Result: Use the .JPG file format.

### **Parameter**

<i>Value</i>	0 The .GIF file format
	1 The .JPG file format
	2 The .PNG file format

## **{SaveHtml.Header}**

### **Syntax**

{SaveHtml.Header *Header*}

### **PerfectScript Syntax**

SaveHtml\_Header (Header?: String)

### **Description**

Lets you specify the text for the header section of the .HTML document.

### **Example**

```
SaveHtml.Header "Header text of this file"
```

### **Parameter**

*Header*      The text

## **{SaveHtml.HeaderDescription}**

### **Syntax**

{SaveHtml.HeaderDescription *HdrDesc*}

### **PerfectScript Syntax**

SaveHtml\_HeaderDescription (HdrDesc?: String)

### **Description**

Lets you specify the header description.

### **Example**

```
SaveHtml.HeaderDescription "Header description for this file"
```

### **Parameter**

*HdrDesc*      The header description

## **{SaveHtml.Layout}**

### **Syntax**

{SaveHtml.Layout *Value*}

### **PerfectScript Syntax**

SaveHtml\_Layout (Value?: Numeric)

### **Description**

Lets you specify the layout to be used.

### **Example**

```
SaveHtml.Layout 2
```

### **Parameter**

*Value*      0 Single page  
              1 Frame enhanced pages  
              2 Multiple pages

## **{SaveHtml.LineBeforeFooter}**

### **Syntax**

{SaveHtml.LineBeforeFooter *Enable*}

### **PerfectScript Syntax**

SaveHtml\_LineBeforeFooter (Enable?: Boolean)

### **Description**

Lets you specify whether or not to insert a line before the footer.

### **Example**

```
SaveHtml.LineBeforeFooter 1
```

### **Parameter**

*Enable*      0 No footer line  
              1 Footer line

► **Related topics**

## **{SaveHtml.LineBeforeHeader}**

### **Syntax**

{SaveHtml.LineBeforeHeader *Enable*}

### **PerfectScript Syntax**

SaveHtml\_LineBeforeHeader (Enable?: Boolean)

### **Description**

Lets you specify whether or not to insert a line before the header.

### **Example**

```
SaveHtml.LineBeforeHeader 0
```

### **Parameter**

<i>Enable</i>	0 No header line
	1 Header line

### **▶ Related topics**

## **{SaveHtml.LinkColor}**

### **Syntax**

{SaveHtml.LinkColor *LinkColor*}

### **PerfectScript Syntax**

SaveHtml\_LinkColor (LinkColor?: String)

### **Description**

Lets you specify the default color of the links

### **Example**

```
SaveHtml.LinkColor Black
```

```
SaveHtml.LinkColor "#ff00ff"
```

### **Parameter**

*LinkColor*      The name of the default link color

#### **► Note**

- SaveHtml.LinkColor will be effective only when the [SaveHtml.UseBrowserColor](#) command is called with 0

## **{SaveHtml.OutputFile}**

### **Syntax**

{SaveHtml.OutputFile *Filename*}

### **PerfectScript Syntax**

SaveHtml\_OutputFile (Filename?: String)

### **Description**

The name of the .HTML file into which the data is to be published.

### **Example**

```
SaveHtml.OutputFile "C:\Shared\New.HTM"
```

### **Parameter**

*Filename*      The name of the .HTML file

### **▶ Related topics**

## **{SaveHtml.OutputType}**

### **Syntax**

{SaveHtml.OutputType *Value*}

### **PerfectScript Syntax**

SaveHtml\_OutputType (Value?: Numeric)

### **Description**

Lets you specify the type of the output file.

### **Example**

```
SaveHtml.OutputType 1
```

### **Parameter**

<i>Value</i>	0 Output as .HTML
	1 Output as .XML
	2 Insert into an existing .HTML file

### **▶ Related topics**



## **{SaveHtml.SaveHtml}**

### **Syntax**

{SaveHtml.SaveHtml}

### **PerfectScript Syntax**

SaveHtml\_SaveHtml

### **Description**

Saves the data from the specified range into a .HTML file. Takes default values if no values have been specified.

## {SaveHtml.Source}

### Syntax

```
{SaveHtml.Source SourceData}
```

### PerfectScript Syntax

```
SaveHtml_OutputFile (SourceData?: String)
```

### Description

Lets you specify the name of the .HTML file into which the data is to be published. *SourceData* consists of two variables delimited by a semicolon.

### Example

```
SaveHtml.Source "Range:A:A11..D11; 0"
```

### Parameters

<i>SourceData</i>	Range
[semicolon delimited]	The range specified as "Range:" followed by page name, followed by actual range.
]	Boolean
	0 Output as table
	1 Output as text

## {SaveHtml.TextColor}

### Syntax

```
{SaveHtml.TextColor TextColor}
```

### PerfectScript Syntax

```
SaveHtml_TextColor (TextColor?: String)
```

### Description

Lets you specify the default color of the text.

### Example

```
SaveHtml.TextColor Black
```

```
SaveHtml.TextColor "#ff00ff"
```

### Parameter

*TextColor* The name of the default text color

#### ► Note

- SaveHtml.TextColor will be effective only when the [SaveHtml.UseBrowserColor](#) command is called with 0

## **{SaveHtml.Title}**

### **Syntax**

{SaveHtml.Title *Title*}

### **PerfectScript Syntax**

SaveHtml\_Title (Title?: String)

### **Description**

Lets you specify the title for the file.

### **Example**

```
SaveHtml.Title "Title info of this file"
```

### **Parameter**

<i>Title</i>	The title for the file
--------------	------------------------

## **{SaveHtml.UseBrowserColor}**

### **Syntax**

{SaveHtml.UseBrowserColor *Enable*}

### **PerfectScript Syntax**

SaveHtml\_UseBrowserColor (Enable?: Boolean)

### **Description**

Lets you specify whether browser colors or the colors you specify are to be used.

### **Example**

```
SaveHtml.UseBrowserColor 0
```

### **Parameter**

<i>Enable</i>	0 Lets you specify the colors. 1 Use the browser's colors.
---------------	---

### **▶ Related topics**

## {SaveHtml.UserDetails}

### Syntax

```
{SaveHtml.UserDetails UserData}
```

### PerfectScript Syntax

```
SaveHtml_UserDetails (UserData?: String)
```

### Description

Lets you specify the user details for the .HTML file. *UserData* consists of three variables delimited by semicolons.

### Example

```
SaveHtml.UserDetails "Sep 23, 1998; Alpha; Alpha@Gamma.com"
```

### Parameters

<i>UserData</i>	Last updated
[semicolon delimited]	Lets you specify the date of the last update.
Updated by	Updated by
]	Lets you specify who last updated the .HTML file.
Email	Email
	Lets you specify an email address.

## {SaveHtml.UseRGBValues}

### Syntax

```
{SaveHtml.UseRGBValues Enable}
```

### PerfectScript Syntax

```
SaveHtml_UseRGBValues (Enable?: Boolean)
```

### Description

Lets you specify whether RGB values are to be used instead of the color name strings.

### Example

```
SaveHtml.UseRGBValues 0
```

### Parameter

<i>Enable</i>	0 Use the color name strings.
	1 Use RGB values.

## {SaveHtml.Wallpaper}

### Syntax

```
{SaveHtml.Wallpaper Filename}
```

### PerfectScript Syntax

```
SaveHtml_Wallpaper (Filename?: String)
```

### Description

Lets you specify which wallpaper is to be used as the background of the .HTML file.

### Example

```
SaveHtml.WallPaper "Stars.JPG"
```

### Parameter

*Filename*      The name of the wallpaper file

## **{Scenario}**

### **Syntax**

*{Scenario.Option}*

### **PerfectScript Syntax**

Scenario\_AddCells ([Block:String])

Scenario\_Capture (ScenarioName:String)

Scenario\_CaptureArea (Area:Enumeration {Notebook!; Page!; Block!; UserDefined!}; [Block:String])

Scenario\_Close ()

Scenario\_DeleteGroup (GroupName:String)

Scenario\_Find ()

Scenario\_Highlight (Highlight?:Enumeration {Yes!; No!}; ChangeCellColor:Numeric; ResultCellColor:Numeric)

Scenario\_NewGroup (GroupName:String)

Scenario\_Open ()

Scenario\_Remove (ScenarioName:String)

Scenario\_RemoveCells ([Block:String])

Scenario\_RenameGroup (OldGroupName:String; NewGroupName:String)

Scenario\_Report (AllGroups?:Enumeration {Yes!; No!}; LeftLabels?:Enumeration {Yes!; No!}; TopLabels?:Enumeration {Yes!; No!}; [Block:String])

Scenario\_Show (ScenarioName:String)

Scenario\_Update\_On\_Block (Update?:Enumeration {Yes!; No!})

Scenario\_UseGroup (GroupName:String)

### **Description**

*{Scenario}* lets you change values in a model, saving the conditions and results for different scenarios. *{Scenario.Open}* must be used prior to using other *{Scenario.Option}* commands; use *{Scenario.Close}* when you are finished using the Scenario Manager. For *{Scenario.AddCells}* and *{Scenario.RemoveCells}*, *Block* defaults to the currently selected cells. For *{Scenario.Report}*, *Block* defaults to the first empty sheet in the notebook.

You can use *{Scenario?}* or *{Scenario!}* to display the Scenario Manager dialog box. *{Scenario?}* lets you manipulate the dialog box, whereas *{Scenario!}* relies on the macro to manipulate it.

### **Example**

The following macro captures the base scenario and two additional scenarios for a car loan.

```
{Scenario.Open}
{Scenario.Capture Base Scenario}
{Scenario.Update}
{SelectBlock A:F4}
{PutCell ".096"}
{SelectBlock A:C5}
{PutCell "60"}
{Scenario.Find}
{Scenario.Highlight 1,7,9}
{Scenario.Capture APR96-60}
{Scenario.Update}
{SelectBlock A:F4}
{PutCell ".085"}
{SelectBlock A:C5}
```

```

{PutCell "48"}
{Scenario.Find}
{Scenario.Highlight 1,7,9}
{Scenario.Capture APR85-48}
{Scenario.Update}
{Scenario.Report 0,1,0}
{FileSaveAs "C:\COREL\SUITE8\DATA\CARS.WB3"}
{Scenario.Close}

```

## Options

{Scenario.AddCells <Block>}	Defines the selected cells as change-and-result cells.
{Scenario.Capture <i>ScenarioName</i> }	Takes a baseline snapshot of data
{Scenario.CaptureArea <i>Area,Block</i> }	Specifies the area where the Scenario Manager tracks data and format changes.
{Scenario.Close}	Closes a Scenario Manager session.
{Scenario.DeleteGroup <i>GroupName</i> }	Deletes the active group and all scenarios in it.
{Scenario.Find}	Automatically locates changed cells after you capture the baseline scenario and make changes.
{Scenario.Highlight <i>Highlight?</i> (0 1), <i>ChangeCellColor</i> (0-15), <i>ResultCellColor</i> (0-15)}	Turns on and off coloring of change-and-result cells.
{Scenario.NewGroup <i>GroupName</i> }	Creates and names a new Scenario Manager group.
{Scenario.Open}	Initializes a Scenario Manager session.
{Scenario.Remove <i>ScenarioName</i> }	Deletes the selected scenario.
{Scenario.RemoveCells <Block>}	Excludes the selected change-and-result cells from the scenario.
{Scenario.RenameGroup <i>OldGroupName,NewGroupName</i> }	Applies another name to the active group.
{Scenario.Report <i>AllGroups</i> (0 1), <i>LeftLabels</i> (0 1), <i>TopLabels</i> (0 1), <Block>}	Creates a summary report of the change-and-result cells in each scenario.
{Scenario.Show <i>ScenarioName</i> }	Lists scenarios you have captured in the active group of scenarios
{Scenario.Update_On_Block <i>Update?</i> (0 1)}	Offers options for using the Scenario Manager to track versions.
{Scenario.UseGroup <i>GroupName</i> }	Lists the scenario groups included in the active notebook.

## {ScenarioExpert}

### Description

{ScenarioExpert} displays the first Scenario Expert dialog box. The macro has no arguments.

### ► Related topics

## **{SCROLLOFF} and {SCROLLON}**

### **Description**

{SCROLLOFF} and {SCROLLON} are equivalent to Scroll Lock off and Scroll Lock on, respectively.

### **▶ Related topics**

# {Search}

## Syntax

{Search.Option}

## PerfectScript Syntax

Search\_Block (Block:String)

Search\_Case (Case:Enumeration {Any!; Exact!})

Search\_Direction (Direction:Enumeration {Row!; Column!})

Search\_Find (String:String)

Search\_Look\_In (LookIn:Enumeration {Formula!; Value!; Condition!})

Search\_Match (Match:Enumeration {Part!; Whole!})

Search\_Next ()

Search\_Previous ()

Search\_Replace ()

Search\_ReplaceAll ()

Search\_ReplaceBy (String:String)

Search\_Reset ()

## Description

{Search} searches for strings in the active sheet. Use {Search.ReplaceBy} to specify the replacement string; {Search.Replace} replaces the string.

You can use {Search?} or {Search!} to display the Find And Replace dialog box. {Search?} lets you manipulate the dialog box, whereas {Search!} relies on the macro to manipulate it.

## Example

The following macro searches the active sheet 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}
```

## Options

{Search.Block <i>Block</i> }	Specifies the cell or multiple cells to search.
{Search.Case Any Exact}	Considers capitalization during the search.
{Search.Direction Column Row}	Searches down columns first, starting with column 1.
{Search.Find <i>String</i> }	Specifies the group of characters to be found in labels, values, and formulas.
{Search.Look_In Condition   Formula   Value}	Specifies what is included in the search.
{Search.Match Part Whole}	Forces the search string to match all of a cell entry.
{Search.Next}	Begins or resumes a forward search without replacing found entries.
{Search.Previous}	Begins or resumes a backward search without replacing found entries.
{Search.Replace}	Lets you decide on an individual basis whether to replace each string found.
{Search.ReplaceAll}	Replaces all found strings without stopping.
{Search.ReplaceBy <i>String</i> }	Specifies the group of characters to substitute for characters found.
{Search.Reset}	Clears any entries in the dialog box and



reinstates the defaults.

## **{SelectAll}**

### **Description**

{SelectAll} selects every cell in the active sheet.

## **{SELECTBLOCK}**

### **Syntax**

{SELECTBLOCK *Block*, <*ActiveCell*>}

### **PerfectScript Syntax**

SelectBlock (Block:String; [ActiveCell:String])

### **Description**

{SELECTBLOCK} lets you select a contiguous or noncontiguous selection within the active notebook. The noncontiguous selections must be enclosed in parentheses.

### **Example**

{SELECTBLOCK A4..B23} selects the cells A4..B23 in the active notebook window.

{SELECTBLOCK (A:A1..A:B12,B:B13..B:C34)} selects the noncontiguous selections A:A1..A:B12, B:B13..B:C34.

### **Parameters**

<i>Block</i>	Coordinates of the cell(s) to select
<i>ActiveCel</i>	Address of the cell within the cells to make active
<i>l</i>	

### **► Related topics**

## **{SELECTFLOAT}**

### **Syntax**

{SELECTFLOAT *ObjectID1*<,*ObjectID2*,...>}

### **PerfectScript Syntax**

SelectFloat (ObjectID:String; {[MoreObjectID:String]})

### **Description**

With {SELECTFLOAT} you can select floating objects in the active notebook window using their names. (To find the name of an object, view it and study its Object Name property.) Use [{SELECTOBJECT}](#) to select objects in a chart or dialog window.

### **Example**

{SELECTFLOAT "Button1"} selects the macro button in the active notebook window with the object name Button1.

### **Parameters**

*ObjectIDx*      Name of the notebook object(s) to select

### **► Related topics**

## **{SELECTOBJECT}**

### **Syntax**

{SELECTOBJECT *ObjectID1*<,*ObjectID2*,...>}

### **PerfectScript Syntax**

SelectObject ([ObjectID:String]; {[MoreObjectID:String]})

### **Description**

With {SELECTOBJECT} you can select objects in the active window using their ID numbers or names. (To find the ID number of an object, view 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 chart window, or icons in the Objects sheet.

### **Example**

{SELECTOBJECT 2,5,7} selects the objects in the active window with the IDs 2, 5, and 7.

### **Parameters**

*ObjectIDx*      Identification number or name of the object(s) to select

### **► Related topics**

## {Series}

### Syntax

{Series.Option}

### PerfectScript Syntax

Series\_Data\_Range (SeriesID:Any; Block:String; [CreatelfNotExist?:Enumeration {Yes!; No!}])

Series\_Delete (SeriesNumber:Numeric; [AndAllSeriesFollowing?:Enumeration {Yes!; No!}])

Series\_Go ()

Series\_Insert (SeriesNumber:Numeric; Block:String)

Series\_Label\_Range (SeriesNumber:Numeric; Block:String; [CreatelfNotExist?:Enumeration {Yes!; No!}])

Series\_Legend (SeriesNumber:Numeric; LegendText:String)

Series\_Reverse\_Series (Yes?:Enumeration {Yes!; No!})

Series\_Swap\_Row\_Col (Yes?:Enumeration {Yes!; No!})

### Description

{Series} creates or deletes chart series.

When you manipulate a series using command equivalents, the changes are not 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 cells that the series should take values from. If you are not sure whether the series exists, set *CreatelfNotExist?* to 1. Then the series will be created if it does not already exist. You can also use {Series.Data\_Range} to set the x-axis series (use "XAxisLabelSeries") or set the legend series (use "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* refers to the cells containing the new series' data.

{Series.Label\_Range} sets up the labels for each value in a series. *Block* refers to the cells containing the labels. If you are not sure whether the series exists, set *CreatelfNotExist?* to 1. Then the series will be created if it does not already exist.

{Series.Legend} sets the legend text for a series (*LegendText* is the new text).

You can use {Series?} or {Series!} to display the Chart Series dialog box. {Series?} lets you manipulate the dialog box, whereas {Series!} relies on the macro to manipulate it.

You can add series to a floating chart using {ADDSERIES}.

### Example

The following macro creates a chart 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}
```

### Options

<pre>{Series.Data_Range SeriesNumber   "AxisLabelSeries"   "LegendSeries", Block &lt;,CreateIfNotExist? (0  1)&gt;}</pre>	<p>Specifies the cell coordinates of the chart data, legend, or label. You must place this value within quotations.</p>
<pre>{Series.Delete SeriesNumber &lt;,AndAllSeriesFollowing? &gt;} {Series.Go}</pre>	<p>Deletes the selected series.</p> <p>Changes the series according to your selections.</p>
<pre>{Series.Insert SeriesNumber, Block} {Series.Label_Range SeriesNumber, Block &lt;,CreateIfNotExist? (0  1)&gt;}</pre>	<p>Adds a new series after the selected series.</p> <p>Specifies the series used for labels.</p>
<pre>{Series.Legend SeriesNumber, LegendText} {Series.Reverse_Series 1  0} {Series.Swap_Row_Col 1  0}</pre>	<p>Specifies the series used for the legend. You must place this value within quotations.</p> <p>Plots the last series first, then moves backwards through the series order.</p> <p>Plots columns as series when Quattro Pro plots series by rows, and plots rows as series when Quattro Pro would plot columns.</p>

## {SeriesManager}

### Syntax

{SeriesManager.Option}

### PerfectScript Syntax

SeriesManager\_Define (Name:String; FormulaOrList:String; FormulaTextOrRepeat:Any; SeedTextOrValue:String; {[Value:String]})

SeriesManager\_Go (Name:String; Orientation:Enumeration {Rows!; Columns!; Tabs!}; [Block:String])

SeriesManager\_Remove (Name:String)

SeriesManager\_Rename (OldName:String; NewName:String)

### Description

{SeriesManager} create a new QuickFill list series. Use {SeriesManager} to create a formula series and a list series.

You can use {SeriesManager?} or {SeriesManager!} to display the Define Fill Series dialog box.

{SeriesManager?} lets you manipulate the dialog box, whereas {SeriesManager!} relies on the macro to manipulate it.

### Example

The following macro creates a SpeedFill series named "First of Month" that consists of the first day of each month in 1995, then fills a column starting at A:A2 with the dates.

```
{SeriesManager.Define "First of Month", List, No, "01/01/95", "02/01/95",
"03/01/95", "04/01/95", "05/01/85", "06/01/95", "07/01/95", "08/01/95",
"09/01/95", "10/01/05", "11/01/95", "12/01/95"}
```

```
{SpeedFill}
```

```
{SeriesManager.Go "First of Month",Columns, A:A2}
```

### Options

<pre>{SeriesManager.Define Name, Formula, FormulaText, SeedText}</pre>	<p>Lets you define a new series.</p>
<pre>{SeriesManager.Define Name, List, Repeating?(0 1),</pre>	<p>Lets you define and name a new series.</p>

*Value1* <,*Value2*,  
*Value3*,...>}  
{SeriesManager.Go *Name*,  
Rows | Columns | Tabs,  
*Block*}  
{SeriesManager.Remove  
*Name*}  
{SeriesManager.Rename  
*OldName*, *NewName*}

Quickly fill cells with a sequence of entries.

Deletes the selected series.

Changes the name for the selected series.

► **Related topics**

## {SetCellString}

### Syntax

```
{SetCellString Cell; String}
```

### PerfectScript Syntax

```
SetCellString (Cell: String; String: String)
```

### Description

Lets you specify the string insert into the cell.

### Example

```
{SetCellString A1; "The string"}
```

### Parameters

<i>Cell</i>	The cell into which you want to insert the string
<i>String</i>	The string you want to insert into the cell

## {SETGRAPHATTR}

### Syntax

```
{SETGRAPHATTR FillColor,BkgColor,FillStyle,BorderColor,BoxType}
```

### PerfectScript Syntax

```
SetGraphAttr (FillColor:String; BkgColor:String; FillStyle:String; BorderColor:String; BoxType:String)
```

### Description

{SETGRAPHATTR} lets you quickly set the properties of all selected objects in the active chart window. If one of the arguments specified in the {SETGRAPHATTR} command is not appropriate for an object, that argument is ignored.

Each color (*FillColor*, *BkgColor*, and *BorderColor*) is in quotes, and specified in RGB format. For *FillStyle*, use any of the strings for that option in the appropriate Object Inspector.

*BoxType* specifies the new border style for the object; use any *Border Style* property string included in a chart Object Inspector.

### Parameters

<i>FillColor</i>	New fill color of the selected object(s)
<i>BkgColor</i>	New background color of the selected object(s)
<i>FillStyle</i>	New fill style of the selected object(s)
<i>BorderCol</i>	New border color of the selected object(s)
<i>or</i>	
<i>BoxType</i>	New border style of the selected object(s)

### ► Related topics



## **{SETLCID}**

### **Syntax**

{SETLCID <LocalID>}

### **PerfectScript Syntax**

SetLCID ([LocalID:Numeric])

### **Description**

{SETLCID} sets the locale ID to the default locale ID or to one specified by *LocalID*. The local ID is a fixed number which specifies language, separator character, and a variety of other international settings; use {SETLCID} to ensure that the automation controller is using the default ID or the ID of a specific target object.

### **Parameters**

<i>LocalID</i>	The value of the local ID
----------------	---------------------------

## {SETMENUBAR}

### Syntax

{SETMENUBAR <SystemDefinition>}

### PerfectScript Syntax

SetMenuBar ([SystemDefinition:String])

### Description

{SETMENUBAR} lets you specify which menu system displays on the menu bar. *SystemDefinition* refers to cells containing the new menu system definition.

You can use {SETMENUBAR} without an argument to restore the default Quattro Pro menu system.

### Example

{SETMENUBAR "A3..C324"} makes the system defined in A3..C324 the active menu system.

### Parameters

<i>SystemDefinition</i>	Cells containing a menu system definition
-------------------------	---

#### ► Note

- This command is obsolete

#### ► Related topics

# {SETOBJECTPROPERTY}

## Syntax

{SETOBJECTPROPERTY *Object.Property,Value*}

## PerfectScript Syntax

SetObjectProperty (ObjectProperty:String; Value:String)

## Description

{SETOBJECTPROPERTY} can change the property settings of many Quattro Pro objects. Selectable objects such as blocks and annotations can also be changed using [{SETPROPERTY}](#). {SETOBJECTPROPERTY} can affect:

**Dialog controls.** Use this syntax to specify a control to manipulate in a dialog window: [*Notebook*]DialogName:ObjectID.Property. [*Notebook*] is optional. 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"}
```

**Chart objects.** Use the same syntax as for dialog controls, but substitute the name of the chart in place of *DialogName*. For example, the following macro changes the size of a rectangle named ColorPick in the chart 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*.. For example, the following macro disables Save in the active menu system:

```
{SETOBJECTPROPERTY "/File/Save.Disabled", "Yes"}
```

## Parameters

*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, this macro copies the text color of the active cells to the text color of A23:

```
{SETOBJECTPROPERTY "A23.Text_Color","Active_Block.Text_Color"}
```

## Parameters

<i>Object</i>	Object to alter property of
<i>Property</i>	Property to alter
<i>Value</i>	New property setting (or another instance of Object.Property to copy the new setting from)

## ▶ [Related topics](#)

## **{SETPOS}**

### **Syntax**

{SETPOS *FilePosition*}

### **Description**

{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 continues in the next row of the macro.

For an example using {SETPOS}, see [{READ}](#).

### **Parameters**

*FilePosition*      the number of bytes into a file to set the file pointer to

### **▶ [Related topics](#)**

## **{SETPROPERTY}**

### **Syntax**

{SETPROPERTY *Property,Value*}

### **PerfectScript Syntax**

SetProperty (Property:String; Value:String)

### **Description**

{SETPROPERTY} alters the properties of the active object (use [{SELECTBLOCK}](#), [{SELECTFLOAT}](#), or [{SELECTOBJECT}](#) to select objects).

To find *Property*, view the object and use the name of the control that sets the property. If the control name is more than one word, connect the words with underscores (\_).

### **Example**

```
{SETPROPERTY "Text_Color", "3"}
```

Result: Sets the selected cells' Text Color property to the fourth color on the notebook palette (the first color is 0).

### **Parameters**

<i>Property</i>	String representing the property to change
<i>Value</i>	String representing the setting to apply to the property

### **▶ Related topics**

## **{SHIFT}**

### **Syntax**

Shift()

### **Description**

This is equivalent to using the shift key.

## **{ ShowErrorMessage }**

### **Syntax**

{ShowErrorMessage}

### **PerfectScript Syntax**

ShowErrorMessage ()

### **Description**

Reinstates the ability for Quattro Pro to show an error message, if one is warranted.

### **▶ Related topics**

## {Slide}

### Syntax

{Slide.Option}

### PerfectScript Syntax

Slide\_Effect (Effect:String)

Slide\_Goto (SlideName:String)

Slide\_Next ()

Slide\_Previous ()

Slide\_Run (SlideShowName:String)

Slide\_Speed (Speed:Numeric)

Slide\_Time (Time:Numeric)

### Description

{Slide} lets you build, edit, and present graphics slide show sequences. *Effect*, *Speed*, and *Time* are the same options offered in the Slide Effect property in the Light Table window. {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 chart button, or attached directly to a QuickButton or custom dialog box button.

### Options

{Slide.Effect <i>Effect</i> }	Specifies the transition effect to use when displaying the next slide in a slide show.
{Slide.Goto <i>SlideName</i> }	Takes the active slide show directly to the slide <i>SlideName</i> .
{Slide.Next}	Advances the active slide show to the next slide.
{Slide.Previous}	Returns the active slide show to the previous slide.
{Slide.Run <i>SlideShowName</i> }	Plays the slide show.
{Slide.Speed 0-15}	Specifies the transition speed to use when displaying the next slide in a slide show.
{Slide.Time <i>Time</i> }	Specifies the time in seconds to display the next slide in a slide show.

## {SlideShowExpert}

### Description

{SlideShowExpert} displays the first Slide Show Expert dialog box. The macro has no arguments.

### ▶ Related topics



## **{SolveFor}**

### **Syntax**

{SolveFor.Option}

### **PerfectScript Syntax**

SolveFor\_Accuracy (Value:Numeric)

SolveFor\_Formula\_Cell (Cell:String)

SolveFor\_Go ()

SolveFor\_Max\_Iters (Iters:Numeric)

SolveFor\_Reset ()

SolveFor\_Target\_Value (Value:Numeric)

SolveFor\_Variable\_Cell (Cell:String)

### **Description**

{SolveFor} solves goal-seeking problems with one variable.

{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.

You can use {SolveFor?} or {SolveFor!} to display the Solve For dialog box. {SolveFor?} lets you manipulate the dialog box, whereas {SolveFor!} relies on the macro to manipulate it.

### **Options**

{SolveFor.Accuracy Value}	Specifies how close Solve For must get to the Target Value.
{SolveFor.Variable_Cell Cell}	Indicates which cell Quattro Pro can change to solve for a desired value.
{SolveFor.Formula_Cell Cell}	Specifies the cell containing the formula you want to solve.
{SolveFor.Go}	Solves for the Target Value.
{SolveFor.Max_Iters Value}	Determines how many passes Solve For makes to solve the formula.
{SolveFor.Reset}	Clears all Solve For settings.
{SolveFor.Target_Value Value}	Specifies the result you want from the Formula Cell.

## **{Sort}**

### **Syntax**

{Sort.Option}

### **PerfectScript Syntax**

Sort\_BlankCellsFirst (BlankFirst?:Enumeration {Yes!; No!})

Sort\_Block (Block:String)

Sort\_Data (Order:String)

Sort\_Go ()

Sort\_Heading (Heading?:Enumeration {Yes!; No!})

Sort\_Key\_1 (Cell:String)

Sort\_Key\_2 (Cell:String)

Sort\_Key\_3 (Cell:String)

Sort\_Key\_4 (Cell:String)

Sort\_Key\_5 (Cell:String)

Sort\_Labels (Use:String)

Sort\_Order\_1 (Order:String)  
 Sort\_Order\_2 (Order:String)  
 Sort\_Order\_3 (Order:String)  
 Sort\_Order\_4 (Order:String)  
 Sort\_Order\_5 (Order:String)  
 Sort\_PreviousSorts (PreviousSorts?:Numeric)  
 Sort\_Reset ()  
 Sort\_Type (Type?:String)

## Description

{Sort} sorts the entries in cells. To perform the sort, use {Sort.Go} after the other sort command equivalents.

You can use {Sort?} or {Sort!} to display the Data Sort dialog box. {Sort?} lets you manipulate the dialog box, whereas {Sort!} relies on the macro to manipulate it.

{Sort.Reset} allows Quattro Pro to automatically determine the sort block, the first sort key, and whether there is a heading row, based on the block surrounding the selected cell, or the selected range.

## Example

The following macro sorts the cells A3..C40 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 "A:A3..C40"}
{Sort.Type Top to bottom}
{Sort.Heading 0}
{Sort.Key_1 a25}
{Sort.Key_2 c23}
{Sort.Order_1 Ascending}
{Sort.Order_2 Ascending}
{Sort.BlankCellsFirst No}
{Sort.Data Numbers First}
{Sort.PreviousSorts -1}
{Sort.Labels Dictionary}
{Sort.Go}
```

## Options


{Sort.BlankCellsFirst 0 1}	Determines whether to filter blank cells to the top during a sort.
{Sort.Block Block}	Specifies cells to be sorted, including row labels but excluding column headings.
{Sort.Data "Labels First" "Numbers First"}	Determines whether to sort Labels or Numbers first.
{Sort.Go}	Performs the sort you specified.
{Sort.Heading 0 1}	Determines whether the first row (or column, depending on sorting based on rows or columns) is used as column headings, or is part of the sort block.
{Sort.Key_1-5 Block}	Specifies up to 5 sort keys, in the order they are to be sorted.
{Sort.Labels "Character Code" "Dictionary"}	Specifies whether text sorts in Dictionary order (ordinary alphabetizing rules) or Character Code order (according to character number—for example, uppercase letters before lowercase). Retained for use with previous Quattro Pro version macros.
{Sort.Order_1-5 Ascending  Descending}	Specifies ascending or descending sort order
{Sort.PreviousSorts N}	Stores up to the last five sorts performed in current file.

{Sort.Reset}  
{Sort.Type "Left to  
Right" | "Top to  
Bottom"

Clears all entries and restores defaults.  
Determines to sort by rows or columns.

## {SPEEDFILL}

### Description

{SPEEDFILL} is equivalent to the QuickFill button  on the Toolbar. It fills the selected cells with sequential data, based on entries in the upper-left portion of the cells.  
To create or modify a series used with QuickFill, use [{SeriesManager.Option}](#).

### ► Related topics

# {SpeedFormat}

## Syntax

{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), <*SubTotals?*(0|1)>}

## PerfectScript Syntax

SpeedFormat (FmtName:String; NumFmt?:Enumeration {Yes!; No!}; Font?:Enumeration {Yes!; No!}; Shading?:Enumeration {Yes!; No!}; TextColor?:Enumeration {Yes!; No!}; Align?:Enumeration {Yes!; No!}; LineDraw?:Enumeration {Yes!; No!}; AutoWidth?:Enumeration {Yes!; No!}; ColHead?:Enumeration {Yes!; No!}; ColTotal?:Enumeration {Yes!; No!}; RowHead?:Enumeration {Yes!; No!}; RowTotal?:Enumeration {Yes!; No!}; [SubTotals?:Enumeration {Yes!; No!}])

## Description

{SpeedFormat} applies the format *FmtName* to the selected cells. The arguments *NumFmt?* through *SubTotals?* each specify a part of the format to apply; use 1 to apply the part or 0 to omit the part.

You can use {SpeedFormat?} or {SpeedFormat!} to display the SpeedFormat dialog box. {SpeedFormat?} lets you manipulate the dialog box, whereas {SpeedFormat!} relies on the macro to manipulate it.

To add or remove formats, use [{SpeedFormat.Option}](#).

## Parameters

<i>FmtName</i>	Name of the format to apply
<i>NumFmt?</i>	1 to apply the numeric format; 0 otherwise
<i>Font?</i>	1 to apply the font; 0 otherwise
<i>Shading?</i>	1 to apply the shading; 0 otherwise
<i>TextColor?</i>	1 to apply the text color; 0 otherwise
<i>Align?</i>	1 to apply the alignment; 0 otherwise
<i>LineDraw?</i>	1 to apply the line drawing; 0 otherwise
<i>AutoWidth?</i>	1 to automatically size the columns; 0 otherwise
<i>ColHead?</i>	1 to apply the column heading format; 0 otherwise
<i>ColTotal?</i>	1 to apply the column total format; 0 otherwise
<i>RowHead?</i>	1 to apply the row heading format; 0 otherwise
<i>RowTotal?</i>	1 to apply the row total format; 0 otherwise
<i>SubTotals?</i>	1 to apply the subtotal format; 0 otherwise

## ► [Related topics](#)

## **{SpeedFormat}**

### **Syntax**

`{SpeedFormat.Option}`

### **PerfectScript Syntax**

`SpeedFormat_Add (Name:String; ExampleBlock:String)`

`SpeedFormat_Remove (Name:String)`

### **Description**

`{SpeedFormat}` adds formats to the SpeedFormat dialog box, or removes them. `{SPEEDFORMAT.Add}` lets you specify a name for the new format and the example cells that define the format. `{SpeedFormat.Remove}` deletes a specified format.

### **Example**

The following macro adds a format named "Strauss" to the SpeedFormat dialog box. The format is based on the example cells A:C10..H25.

```
{SpeedFormat.Add "Strauss",A:C10..H25}
```

### **Options**

<code>{SpeedFormat.Add Name, ExampleBlock}</code>	Creates a new custom format.
<code>{SpeedFormat.Remove Name}</code>	Deletes the active SpeedFormat.

### **▶ Related topics**

# {SPEEDSUM}


## Syntax

{SPEEDSUM *Block*}

## PerfectScript Syntax

SpeedSum ([Block:String])

## Description

{SPEEDSUM} is equivalent to selecting cells and choosing the QuickSum button  from the Toolbar. *Block* includes rows and/or columns to sum, plus adjacent empty cells to hold the results; the default *Block* is the current selection.

## Parameters

*Block*

Coordinates of the cells to sum, including blank cells for results

## ► Related topics

## **{STEP}**

### **Description**

{STEP} is equivalent to the Debug key, Shift+F2.

▶ **Related topics**

## **{STEPOFF}**

### **Description**

{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.

### ▶ **Note**

- This command is obsolete

### ▶ **Related topics**



## {STEPON}

### Description

{STEPON} activates Debug mode, which runs macros one step at a time for debugging. When Quattro Pro encounters a {STEPON} command, it pauses and waits for you 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 executing.

### Example

This 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 do not.

```
_clean_up  {; Save notebook to disk and say goodbye}
  { }{STEPON}
  {FileSave}
  {BRANCH _quit_msg}
```

```
_quit_msg  { }{STEPOFF}
  {QUIT}
```

#### ► Note

- This command is obsolete

#### ► Related topics

## {Subroutine}

### Syntax

{Subroutine <ArgumentList>}

### Description

{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.)

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.

### Example

The following example invokes a subroutine that forces you to verify printing twice before it begins.

```
_print      {GETLABEL "Do you want to 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
```

### Parameters

<i>Subroutine</i>	Name of the subroutine being called (which can be a cell name or a cell address)
<i>ArgumentList</i>	List of one or more arguments to be passed to the specified subroutine (optional)

### ► Related topics

## **{ SuppressErrorValue }**

### **Syntax**

{ SuppressErrorValue }

### **PerfectScript Syntax**

SuppressErrorValue ( )

### **Description**

Suppresses the ability for Quattro Pro to return a specific error value, if one is warranted.

#### **► Note**

- This command is obsolete



# **{TAB}**

## **Syntax**

{TAB <*Number*>}

## **Description**

{TAB}, like [{BIGRIGHT}](#), moves one cell to the right.

When the Compatible Keys option is checked, {TAB} selects the leftmost cell of the screen that is 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.

## **Parameters**

<i>Number</i>	Any positive integer or the address of a cell containing a positive integer (optional)
---------------	--

### ► **Note**

- This command is obsolete

### ► **Related topics**

## **{TabControl}**

### **Syntax**

{TabControl.Option}

### **Description**

{TabControl} lets you add, remove, and rearrange tabs in the current tab control.

### **Options**

{TabControl.Add_Page <i>TabControlName, Position</i> }	Adds the tab indicated in Page Name after the tab name selected in Page List.
{TabControl.Delete_Page <i>TabControlName, TabButtonName</i> }	Removes the selected tab name from the tab control.
{TabControl.Insert_Page <i>TabControlName, Position</i> }	Adds the tab indicated in Page Name before the tab name selected in Page List.
{TabControl.Move_Page <i>TabControlName, TabButtonName,Up Down</i> }	Changes the position of the tab name selected in Page List.
{TabControl.SelectPage <i>TabControlName, TabButtonName</i> }	Selects a named sheet.

### **► Note**

- This command is obsolete

## **{TABLE}**

### **Description**

{TABLE} repeats the last What-If operation.

▶ **Related topics**

## {TableLink}

### Syntax

{TableLink.Option}

### PerfectScript Syntax

TableLink\_Block (Block:String)

TableLink\_Go ()

TableLink\_Name (TableName:String)

### Description

{TableLink} establishes a link to an external database table and displays the table in a Quattro Pro notebook.

You can use {TableLink?} or {TableLink!} to display the Table Link dialog box. {TableLink?} lets you manipulate the dialog box, whereas {TableLink!} relies on the macro to manipulate it.

### Options

{TableLink.Block Block}	Specifies the cells where you want the linked table to appear.
{TableLink.Name e TableName}	Sets the filename of the database table to which you want to establish a link.
{TableLink.Go}	Links the table.

## {TableQuery}

### Syntax

{TableQuery.Option}

### PerfectScript Syntax

TableQuery\_Destination (Block:String)

TableQuery\_FileQuery (Yes?:Enumeration {Yes!; No!})

TableQuery\_Go ()

TableQuery\_QueryBlock (Block:String)

TableQuery\_QueryFile (Filename:String)

### Description

{TableQuery} lets you search external databases for records. The query is not performed until {TableQuery.Go} is used.

You can use {TableQuery?} or {TableQuery!} to display the Table Query dialog box. {TableQuery?} lets you manipulate the dialog box, whereas {TableQuery!} relies on the macro to manipulate it.

### Example

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 cell task\_query.

```
{TableQuery.FileQuery No}
{TableQuery.QueryBlock task_query}
{TableQuery.Destination A:A2}
{TableQuery.Go}
```

### Options

{TableQuery.Destina	Specifies the cells for the query's Answer Table (its
---------------------	---



tion *Block*}  
{TableQuery.FileQuery Yes|No}  
{TableQuery.Go}  
{TableQuery.QueryBlock *Block*}  
{TableQuery.QueryFile *Filename*}

results).  
Specifies an external query file as the source of the query text.  
Performs the table query.  
Specifies cells in the active notebook as the source of the query text.  
Specifies the filename or cell coordinates of the query text.

► **Related topics**

## **{TableView}**

### **Description**

{TableView} launches the Database Desktop.

## {TemplateTB}

### Syntax

{TemplateTB.Option}

### PerfectScript Syntax

TemplateTB\_Add (Name:String; Path:String)

TemplateTB\_Context (Name:String; Settings:String)

TemplateTB\_Docking\_Position (Name:String; Position:Enumeration {Top!; Bottom!; Left!; Right!; Floating!}; [Context:Numeric])

TemplateTB\_Hide (Name:String)

TemplateTB\_Remove (Name:String)

TemplateTB\_Rename (Name:String; NewName:String)

TemplateTB\_Reset (Name:String)

TemplateTB\_Show (Name:String)

### Description

{TemplateTB} is similar to {Toolbar.Option} except that it controls the Template toolbar.

### Options

{TemplateTB.Add Name, Path}	Adds a new Template toolbar.
{TemplateTB.Show Name}	Shows a Template toolbar.
{TemplateTB.Hide Name}	Hides a Template toolbar.
{TemplateTB.Remove Name}	Removes a Template toolbar.
{TemplateTB.Reset Name}	Resets a Template toolbar to its default setup.
{TemplateTB.Docking_Position Name, Top   Left   Right   Bottom   Floating}	Sets the docking position of a Template toolbar.
{TemplateTB.Rename Name, NewName}	Renames a Template toolbar.
{TemplateTB.Context Name, Desktop (Yes   No), Notebook (Yes   No), Chart (Yes   No), Dialog (Yes   No), Objects Page (Yes   No), Slide Show (Yes   No)}	Sets the contexts in Quattro Pro in which a Template toolbar appears.

## {TERMINATE}

### Syntax

{TERMINATE DDEChannel}

### Description

{TERMINATE} closes down a DDE conversation opened with [{INITIATE}](#).

For an example using {TERMINATE}, see [{EXECUTE}](#).

### Parameters

DDEChanne  
l Channel number of the DDE conversation to terminate

#### ► Note

- This command is obsolete

#### ► [Related topics](#)

## {Toolbar}

### Syntax

{Toolbar.Option}

### PerfectScript Syntax

Toolbar\_Add (Name:String; Path:String)

Toolbar\_Context (Name:String; Settings:String)

Toolbar\_Docking\_Position (Name:String; Position:Enumeration {Top!; Bottom!; Left!; Right!; Floating!}; [Context:Numeric])

Toolbar\_Hide (Name:String)

Toolbar\_Remove (Name:String)

Toolbar\_Rename (Name:String; NewName:String)

Toolbar\_Reset (Name:String)

Toolbar\_Show (Name:String)

### Description

{Toolbar} displays and hides toolbars.

{Toolbar.Docking\_Position Order} is a numeric number used to position the selected toolbar in relation to other visible toolbars at the specified docking position:

-1	Displays the specified toolbar at the end of toolbars at the specified docking position
0	Displays the specified toolbar at the beginning of toolbars at the specified docking position
1	Displays the specified toolbar 1 position in from the beginning of toolbars at the specified docking position
n	Displays the specified toolbar n positions in from the beginning of toolbars at the specified docking position.

To record such macros as adding, positioning, and removing toolbars, right-click anywhere on a visible toolbar while recording a macro.

### Options

{Toolbar.Add Name, Path}	Lets you create a toolbar and add it to the toolbar list.
{Toolbar.Context Name, Desktop(1 0), Notebook(1 0), Chart(1 0), Dialog(1 0), Objects Page(1 0), Slide Show(1 0)}	Lists all toolbars. <b>The option is obsolete.</b>
{Toolbar.Docking_Position Name, Top   Left   Right   Bottom   Floating, Order}	Specifies where the toolbar will appear when on screen.
{Toolbar.Hide Name}	Hides the selected toolbar.
{Toolbar.Remove Name}	Removes a toolbar you created from the list.
{Toolbar.Rename Name, NewName}	Specifies the name of the toolbar.
{Toolbar.Reset Name}	Changes the selected standard Quattro Pro toolbar back to its default settings.
{Toolbar.Show Name}	Displays the selected toolbar.

#### ► Note

- This command is obsolete

## **{TTESTM}**

### **Syntax**

{TTESTM *InBlock1*,*InBlock2*,*OutBlock*,<*Labels*>,<*Alpha*>,<*Difference*>}

### **Description**

{TTESTM} performs a Student's t-Test using two independent (rather than paired) samples with equal variances. {TTESTM} is equivalent to the t-Test analysis tool.

### **Parameters**

<i>InBlock1</i>	One or more numeric cell values representing the first input cells
<i>InBlock2</i>	One or more numeric cell values representing the second input cells
<i>OutBlock</i>	Upper-left cell of the output cells
<i>Labels</i>	1 if labels are located in the first column or row of the input cells; 0 if the input cells do not contain labels; the default is 0
<i>Alpha</i>	Significance level of the test; the default is 0.05
<i>Difference</i>	Value indicating the hypothetical difference in the means between <i>InBlock1</i> and <i>InBlock2</i> ; the default is 0
<i>e</i>	

#### **► Note**

- This command is obsolete

#### **► Related topics**



## **{UNDO}**

### **Description**

{UNDO} "takes back" the last command given and restores the previous state for most commands.

▶ **Related topics**

## **{UngroupObjects}**

### **Description**

{UngroupObjects} separates the selected group of chart annotation objects so each can be moved or modified without affecting the others.



## {UP} and {U}

### Syntax

{UP <Number>} or {U <Number>}

### Description

{UP} and {U} are equivalent to the Up Arrow key. The optional argument *Number* moves the selector up the corresponding number of rows. You can use cell references or cell names as arguments.

### Example

{UP}{UP}{UP}	Moves the selector up three rows
{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 selection named temp

### Parameters

*Number* Any positive integer (optional)

#### ► Note

- This command is obsolete

#### ► Related topics

## **{UIWorkspace.GetCurrent}**

### **Syntax**

{UIWorkspace.Get Current *celladdress*}

### **PerfectScript Syntax**

None

### **Description**

{UIWorkspace.GetCurrent *celladdress*} places the name of the current workspace in the specified Cell address.

## **{UIWorkspace.Exists}**

### **Syntax**

{UIWorkspace.Exists *workspace name*, *celladdress*}

### **PerfectScript Syntax**

None

### **Description**

{UIWorkspace.Exists *workspace name*, *celladdress*} checks to see if the specified workspace name currently exists.

## **{UIWorkspace.Import}**

### **Syntax**

{UIWorkspace.Import *filename*, *workspace name*, <*description*>, <*workspace base*>}

### **PerfectScript Syntax**

None

### **Description**

{UIWorkspace.Import *filename*, *workspace name*, <*description*>, <*workspace base*>} imports a workspace file (\*.cwf).

### **Parameters**

<i>filename</i>	Filename and path of .cwf file
<i>workspace name</i>	Name of workspace
<i>description</i>	Optional parameter which allows you to enter a description of the workspace. If not specified, the descriptions default value is blank.
<i>workspace base</i>	Optional parameter which specifies the workspace on which to base this new workspace. If not specified, it is assumed that it is based on the "_default" workspace.

## **{UIWorkspace.Load}**

### **Syntax**

{UIWorkspace.Load *workspace name*}

### **PerfectScript Syntax**

None

### **Description**

{UIWorkspace.Load *workspace name*} sets the specified workspace as the current workspace.

## **{VLINE}**

### **Syntax**

{VLINE *Distance*}

### **PerfectScript Syntax**

VLine (Distance:Numeric)

### **Description**

{VLINE} scrolls the active notebook vertically by *Distance* rows. If the number is positive, it scrolls down; if negative, it scrolls up. {VLINE} does not move the selector; only the view of the notebook is altered. Use [{DOWN}](#) or [{UP}](#) to move the selector vertically.

### **Example**

{VLINE 11} scrolls the display 11 rows down.

{VLINE -4} scrolls the display 4 columns up.

### **Parameters**

*Distance*      Number of rows to scroll the active notebook vertically

### **▶ Related topics**

## **{VPAGE}**

### **Syntax**

{VPAGE *Distance*}

### **PerfectScript Syntax**

VPage (Distance:Numeric)

### **Description**

{VPAGE} scrolls the active notebook vertically by *Distance* screens. If the number is positive, it scrolls down; if negative, it scrolls up. {VPAGE} does not move the selector; only the view of the notebook is altered. Use the commands [{PGDN}](#) or [{PGUP}](#) to move the selector vertically.

### **Parameters**

*Distance*      Number of screens to scroll the active notebook vertically

### **▶ [Related topics](#)**





## **{WAIT}**

### **Syntax**

`{WAIT DateTimeNumber}`

### **Description**

`{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.

### **Example**

The following macro beeps, displays a "Go home" message, and suspends all execution until 8:00 a.m. the next day:

```
{BEEP 3}
{PUTCELL "Time to go home!"}
{DOWN}
{WAIT @TODAY+1+@TIMEVALUE("8:00 AM")}
```

This macro waits for one day:

```
{WAIT @NOW+1}
```

This macro uses `{WAIT}` to alert you 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
```

### **Parameters**

<i>DateTimeNumber</i>	The date and time at which macro execution can resume
-----------------------	---

#### **► Note**

- This command is obsolete

#### **► Related topics**

## **{WaitCursorOn} and {WaitCursorOff}**

### **Description**

{WaitCursorOn} and {WaitCursorOff} turn on or off the default Windows "busy" cursor.

### **► Note**

- This command is obsolete

## {WebQuery.Create}

### Syntax

```
{WebQuery.Create Filedata}
```

### PerfectScript Syntax

```
WebQuery_Create (Filedata?: String)
```

### Description

Lets you create a new Web query file.

### Parameter

<i>Filedata</i>	The name of the new query file
-----------------	--------------------------------

## {WebQuery.Destination}

### Syntax

```
{WebQuery.Destination DestRange}
```

### PerfectScript Syntax

```
WebQuery_Destination (DestRange?: String)
```

### Description

Lets you specify the output location. If empty, a new sheet will be used.

### Parameter

<i>DestRange</i>	The range of cells
------------------	--------------------

## {WebQuery.EditLink}

### Syntax

```
{WebQuery.EditLink}
```

### PerfectScript Syntax

```
WebQuery_EditLink ()
```

### Description

Lets you edit the web link that is stored in the selected cell.

### Example

```
WebQuery_InitLink
```

```
WebQuery_LinkRefreshType "0;1"
```

```
WebQuery_EditLink
```

#### ► **Note**

- This command should be preceded by a set of property commands.

#### ► **Related topics**

## **{WebQuery.InitLink}**

### **Syntax**

{WebQuery.InitLink}

### **PerfectScript Syntax**

WebQuery\_InitLink ()

### **Description**

Lets you initialize the Web link.

#### **► Note**

- This is mandatory command that does the necessary initialization for editing the link.

## **{WebQuery.LinkRange}**

### **Syntax**

```
{WebQuery.LinkRange LinkRange}
```

### **PerfectScript Syntax**

```
WebQuery_LinkRange (LinkRange?: String)
```

### **Description**

Lets you specify the range of cells to be associated with Web link.

### **Parameter**

<i>LinkRange</i>	The range of cells
------------------	--------------------

## **{WebQuery.LinkRefreshDuration}**

### **Syntax**

```
{WebQuery.LinkRefreshDuration Value}
```

### **PerfectScript Syntax**

```
WebQuery_LinkRefreshDuration (Value?: String)
```

### **Description**

Lets you specify the refresh duration in seconds

### **Parameter**

<i>Value</i>	Must be in the format "hh:mm:ss"
--------------	----------------------------------

## **{WebQuery.LinkRefreshTime}**

### **Syntax**

```
{WebQuery.LinkRefreshTime Time}
```

### **PerfectScript Syntax**

```
WebQuery_LinkRefreshTime (Time?: String)
```

### **Description**

Lets you specify the value of start time, end time, start day, and end day.

### **Parameter**

<i>Time</i>	Must be in the format "hh:mm:ss"
-------------	----------------------------------

## **{WebQuery.LinkRefreshType}**

### **Syntax**

```
{WebQuery.LinkRefreshType LinkRefreshOptions}
```

### **PerfectScript Syntax**

```
WebQuery_LinkRefreshType (LinkRefreshOptions?: String)
```

### **Description**

Lets you specify the refresh options. *LinkRefreshOptions* consists of two variables delimited by a semicolon.

### **Example**

```
{WebQuery.LinkRefreshType 2; 0}
```

### **Parameters**

<i>LinkRefreshOptions</i> [semicolon delimited]	Refresh Type 0 Duration 1 Start time 2 End time 3 Start day 4 End day
	Boolean 0 False 1 True

## {WebQuery.LinkWrapOption}

### Syntax

```
{WebQuery.LinkWrapOption LinkWrapOptions}
```

### PerfectScript Syntax

```
WebQuery_LinkWrapOption (LinkWrapOptions?: String)
```

### Description

Lets you specify the wrap options. *LinkWrapOptions* consists of two variables delimited by a semicolon.

### Example

```
{WebQuery.LinkWrapOption 0; 1}
```

### Parameters

<i>LinkWrapOptions</i> [semicolon delimited]	Wrap Type 0 Wrap at the beginning of range 1 Insert data at the end of range 2 Insert data at the beginning of range
	Boolean (Can be TRUE/FALSE only when WrapType is 0). 0 False 1 True

## {WebQuery.QueryFileName}

### Syntax

```
{WebQuery.QueryFileName FileName}
```

### PerfectScript Syntax

```
WebQuery_QueryFileName (FileName?: String)
```

### Description

Lets you specify the query file to be used.

### Parameter

<i>FileName</i>	The name of the query file.
-----------------	-----------------------------

## {WebQuery.RefreshWebLink}

### Syntax

```
{WebQuery.RefreshWebLink}
```

### PerfectScript Syntax

```
WebQuery_RefreshWebLink ()
```

### Description

Lets you refresh the current web link.

► **Note**

- There is a specification issue regarding the RefreshWebLink functionality. There is no way to run the link and download the data unconditionally. The user has to change the start time/date etc. or the system time to force the immediate update. It will be very useful to the user if there is any "Update link now" command.

## **{WebQuery.Run}**

### **Syntax**

{WebQuery.Run}

### **PerfectScript Syntax**

WebQuery\_Run ()

### **Description**

Lets you run the current query.



## {WebQuery.SetQueryOptions}

### Syntax

```
{WebQuery.SetQueryOptions QueryOpts}
```

### PerfectScript Syntax

```
WebQuery_SetQueryOptions (QueryOpts?: String)
```

### Description

Lets you specify the values of the query options. *QueryOpts* consists of two variables delimited by a semicolon.

### Example

```
{WebQuery.SetQueryOptions 1; 1}
```

### Parameters

<i>QueryOpts</i> [semicolon delimited]	Type
	0 Save as Web Link
	1 Import only tables
	2 Auto-size
	3 Retain HTML format
	4 Refresh on open
	Boolean
	0 False
	1 True

## {WebQuery.SetQueryParameters}

### Syntax

```
{WebQuery.SetQueryParameters QueryParams}
```

### PerfectScript Syntax

```
WebQuery_SetQueryParameters (QueryParams?: String)
```

### Description

Lets you specify the parameter value options.

### Parameters

<i>QueryParams</i> [semicolon delimited]	Parameter
	Parameter type
	Parameter value

## {WebQuery.Source}

### Syntax

```
{WebQuery.Source SourceRange}
```

### PerfectScript Syntax

```
WebQuery_Source (SourceRange?: String)
```

### Description

Lets you specify the range of cells to be updated from the query output.

### Parameter

<i>Range</i>	The range of cells
--------------	--------------------

## {WhatIf}

## Syntax

{WhatIf.Option}

## PerfectScript Syntax

WhatIf\_Block (Block:String)

WhatIf\_Input\_Cell\_1 (Cell:String)

WhatIf\_Input\_Cell\_2 (Cell:String)

WhatIf\_One\_Way ()

WhatIf\_Reset ()

WhatIf\_Two\_Way ()

## Description

{WhatIf} builds one- or two-variable "what-if" tables that display a range of results for different conditions.

If you are 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}.

You can use {WhatIf?} or {WhatIf!} to display the What-If dialog box. {WhatIf?} lets you manipulate the dialog box, whereas {WhatIf!} relies on the macro to manipulate it.

## Example

The following macro defines A4..H18 as the "what-if" cells, 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}
```

## Options

{WhatIf.Block <i>Block</i> }	Specifies the cells where you want to write the data table.
{WhatIf.Input_Cell_1 <i>Cell</i> }	Specifies the first (or only) cell referenced by the what-if formula.
{WhatIf.Input_Cell_2 <i>Cell</i> }	Specifies the second cell referenced by a two-variable what-if formula.
{WhatIf.One_Way}	Builds the table.
{WhatIf.Reset}	Clears all settings.
{WhatIf.Two_Way}	Builds the table.

## {WhatIfExpert}

### Description

{WhatIfExpert} displays the first What-If Expert dialog box. The macro has no arguments

### ► Related topics

## **{WINDOW}**

### **Syntax**

{WINDOW<*Number*>}

### **Description**

{WINDOW} switches to the specified open notebook window (1-9). This command is for compatibility with Quattro Pro for DOS; use {ACTIVATE} to activate windows when developing macros for Quattro Pro for Windows. The argument *Number* is optional; {WINDOW} without an argument is equivalent to the Pane key, F6.

### **Parameters**

<i>Number</i>	Number of an open notebook window (1-9)
---------------	---

#### ► **Note**

- This command is obsolete

#### ► **Related topics**

## **{WindowArrIcon}**

### **Description**

{WindowArrIcon} lines up minimized windows on the Quattro Pro desktop or icons on the Objects sheet.

## **{WindowCascade}**

### **Description**

{WindowCascade} rearranges all open windows on the Quattro Pro desktop.

## **{WindowClose}**

### **Description**

{WindowClose} is equivalent to Close in a Control menu, which closes the active window (if the active window is not saved, a prompt appears to confirm the operation).

### **▶ Related topics**

## **{WindowHide}**

### **Description**

{WindowHide} conceals the active notebook window.

## **{WindowMaximize}**

### **Description**

{WindowMaximize} is equivalent to Maximize in a Control menu, which enlarges the active window so it fills the screen.



## **{WindowMinimize}**

### **Description**

{WindowMinimize} is equivalent to Minimize in a Control menu, which shrinks the active window to an icon on the Quattro Pro desktop.

## **{WindowMove}**

### **Syntax**

{WindowMove *UpperLeftX*, *UpperLeftY*}

### **PerfectScript Syntax**

WindowMove (*UpperLeftX*:Numeric; *UpperLeftY*:Numeric)

### **Description**

{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.

### **Parameters**

<i>UpperLeft</i>	Distance between the left side of the Quattro Pro window and the
<i>X</i>	left side of the active window, in pixels
<i>UpperLeft</i>	Distance between the bottom of the input line and the top of the
<i>Y</i>	active window, in pixels

## **{WindowNewView}**

### **Description**

{WindowNewView} displays a duplicate copy of the active notebook in a new window.

## **{WindowNext}**

### **Description**

{WindowNext} is equivalent to choosing Next in a Control menu or pressing Ctrl+F6. It makes the next window active.

## {WindowPanes}

### Syntax

{WindowPanes Horizontal|Vertical|Clear, *Synch?*(0|1), <Width>, <Height>}

### PerfectScript Syntax

WindowPanes (Mode:Enumeration {Clear!; Horizontal!; Vertical!}; *Synch?*:Enumeration {Yes!; No!}; [Width:Numeric]; [Height:Numeric])

### Description

{WindowPanes} splits a notebook window into two horizontal or vertical panes; use Clear to restore a single pane.

*Width* and *Height* indicate the ratio relationship between the panes.

You can use {WindowPanes?} or {WindowPanes!} to display the Split Window dialog box. {WindowPanes?} lets you manipulate the dialog box, whereas {WindowPanes!} relies on the macro to manipulate it.

### Example

{WindowPanes Vertical,0,2,1} splits the notebook window into two vertical panes, not synchronized. The first pane is twice as wide as the second.

### Parameters

<i>Synch?</i>	Whether the panes are synchronized: yes (1) or no (0)
<i>Width</i>	Width of the left pane or height of the upper pane (optional)
<i>Height</i>	Width of the right pane or height of the lower pane (optional)

## {WindowQPW}

### Syntax

{WindowQPW.*Option*}

### PerfectScript Syntax

WindowQPW\_Maximize ()

WindowQPW\_Minimize ()

WindowQPW\_Restore ()

### Description

{WindowQPW} is the command equivalent for the Maximize, Minimize, and Restore commands on the Quattro Pro Control menu.

{WindowQPW.Maximize} enlarges the Quattro Pro window so it fills the screen.

{WindowQPW.Minimize} shrinks the Quattro Pro window to an icon.

{WindowQPW.Restore} restores the Quattro Pro window to its original size.

### Options

{WindowQPW.Maximize}	Maximizes the Quattro Pro application window.
{WindowQPW.Minimize}	Minimizes the Quattro Pro application window.
{WindowQPW.Restore}	Restores the Quattro Pro application window to its previous size.

### ► Related topics

## **{WindowRestore}**

### **Description**

{WindowRestore} is equivalent to Restore on the Control menu. It restores minimized windows to their original size.

## **{WindowShow}**

### **Syntax**

{WindowShow *Name*}

### **PerfectScript Syntax**

WindowShow (Name:String)

### **Description**

{WindowShow} shows hidden window *Name* and makes it active.

You can use {WindowShow?} or {WindowShow!} to display the Show Window dialog box. {WindowShow?} lets you manipulate the dialog box, whereas {WindowShow!} relies on the macro to manipulate it.

### **Parameters**

<i>Name</i>	Name of the hidden window to show
-------------	-----------------------------------

## **{WindowSize}**

### **Syntax**

{WindowSize *X*,*Y*}

### **PerfectScript Syntax**

WindowSize (*X*:Numeric; *Y*:Numeric)

### **Description**

{WindowSize} is equivalent to Size in the Control menu. It sizes the active window to the specified width and height.

### **Parameters**

<i>X</i>	New window width, in pixels
<i>Y</i>	New window height, in pixels

## **{WINDOWSOFF}**

### **Description**

{WINDOWSOFF} disables normal screen updating during macro execution when Quattro Pro's Macro Suppress-Redraw property is set to None. It can significantly 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 executing, so you are not "locked out" of the screen. To cancel its effect within the same macro, use {WINDOWSON}.

Use {WINDOWSOFF} with {PANELOFF} to completely disable normal screen updating.

After a {WINDOWSOFF} command, avoid pointing to cells in response to an Edit command. The selector may be in a different cell than the "frozen" display indicates. If you must point to cells, precede it with a {WINDOWSON} command.

### **Example**

The following macro uses {WINDOWSOFF} and {WINDOWSON} to turn off screen updating while Quattro Pro sorts a list of vendors with the cell 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_nameGeneral Cement Co.
  Alveoli Mfg., Inc.
  Sandab Development
  Consolidated Dust
```

► **Related topics**

## **{WINDOWSON}**

### **Description**

{WINDOWSON} reenables normal screen updating during macro execution, canceling the effects of a previous [{WINDOWSOFF}](#). However, the screen will not be updated until [{CALC}](#) is encountered or the macro ends. If {WINDOWSON} is called when screen updating is already in effect, the command is ignored.

See [{WINDOWSOFF}](#) for an example using {WINDOWSON}.

### **▶ [Related topics](#)**



## **{WindowTile}**

### **Description**

{WindowTile} displays all open windows without overlapping them.

## **{WindowTile.TileTopToBottom}**

### **Description**

{WindowTile.TileTopToBottom} tiles multiple files horizontally.

## **{WindowTitles}**

### **Syntax**

{WindowTitles Horizontal | Vertical | Both | Clear}

### **PerfectScript Syntax**

WindowTitles (Mode:Enumeration {Clear!; Horizontal!; Vertical!; Both!})

### **Description**

{WindowTitles} locks specific rows and/or columns of a spreadsheet sheet as titles on screen. When you scroll, the titles remain fixed on screen 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.

You can use {WindowTitles?} or {WindowTitles!} to display the Locked Titles dialog box. {WindowTitles?} lets you manipulate the dialog box, whereas {WindowTitles!} relies on the macro to manipulate it.

You can use {WindowTitles.Title} with @COMMAND, @PROPERTY, and @CURVALUE.

### **Example**

Use @COMMAND{"WindowTitles.Title"} to determine whether locked titles are in use and to display their type (Horizontal, Vertical, Both, or Clear). You can also use this command in macros to check for locked titles.

```
\A{Calc} {If TitlesOn} {WindowTitles Clear} {Quit}  
{WindowTitles Both}  
TitlesOn @COMMAND("WindowTitles.Title") = "Both"
```

### **► Related topics**

## **{Workflow.RouteDocument}**

### **Syntax**

{Workflow.RouteDocument *Filename*}

### **PerfectScript Syntax**

Workflow\_RouteDocument (Filename: String)

### **Description**

#### **Parameter**

*Filename*            The name of the document you want to route

## **{Workflow.WorkflowManager}**

### **Syntax**

{Workflow.WorkflowManager}

### **PerfectScript Syntax**

Workflow\_WorkflowManager ()

## {Workspace}

### Syntax

{Workspace.Option}

### PerfectScript Syntax

Workspace\_Restore (Filename:String)

Workspace\_Save (Filename:String)

### Description

{Workspace.Save} saves all open notebooks as a group with the specified *Filename* (Quattro Pro's default file extension for workspaces is .WBS). {Workspace.Restore} opens the specified file.

### Options

{Workspace.Restore <i>Filename</i> }	Overlays any existing windows with the windows stored in the workspace file, then retrieves the appropriate file for each.
{Workspace.Save <i>Filename</i> }	Saves the position and size of all notebook windows and the names of the files contained in each window.

## {WRITE}

### Syntax

{WRITE String<,String2,String3,...>}

### Description

{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. See [{GETPOS}](#) for a discussion of the file pointer.

{WRITE} does not 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*, 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 is 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 does not contain a label, an error message displays.

### Example

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}
```

## Parameters

*String* String of characters to be written into the open file

### ► **Note**

- After you finish accessing a file with {WRITE}, it **must** be closed with {CLOSE}. Otherwise, the file could become corrupted if the computer is turned off or otherwise interrupted.
- This command is obsolete

### ► **Related topics**

# {WRITELN}

## Syntax

{WRITELN String<,String2,String3,...>}

## Description

{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. (See [{GETPOS}](#) for a discussion of the file pointer.)

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*, separate them with commas. For example, to write the label Dear Ms. followed by the contents of B6 into the file, you would use

```
{WRITELN "Dear Ms. ",+B6}
```

A carriage return and linefeed character are 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 is insufficient room on the disk to increase a 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.

## Example

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}
```

## Parameters

*String*           String of characters to be written into the open file as a single line

### ► Note

- After you finish accessing a file with {WRITELN}, you **must** close the file with [{CLOSE}](#). Otherwise, the file could become corrupted if the computer is turned off or otherwise interrupted.
- This command is obsolete

### ► [Related topics](#)





## {XMLTag.AutoGenerate}

### Syntax

{XMLTag.AutoGenerate *Block*; *LabelsTop*; *LabelsLeft*; *LabelsBottom*; *LabelsRight*; *Intersection*}

### PerfectScript Syntax

XMLTag\_AutoGenerate (Block: String; LabelsTop: Boolean; LabelsLeft: Boolean; LabelsBottom: Boolean; LabelsRight: Boolean)

### Description

Equivalent to Insert ► XML Tag...  
► Generate...

### Parameters

<i>Block</i>	The Block
<i>LabelsTop</i>	0
	1
<i>LabelsLef</i>	0
<i>t</i>	1
<i>LabelsBot</i>	0
<i>tom</i>	1
<i>LabelsRig</i>	0
<i>ht</i>	1
<i>Intersecti</i>	0
<i>on</i>	1

## {XMLTag.Create}

### Syntax

{XMLTag.Create *TagName*; *Block*}

### PerfectScript Syntax

XMLTag\_Create (TagName: String; Block: String)

### Description

Equivalent to Insert ► XML Tag...  
► Generate...

### Parameters

<i>TagName</i>
<i>Block</i>

## {XMLTag.Delete}

### Syntax

{XMLTag.Delete *TagName*}

### PerfectScript Syntax

XMLTag\_Delete (TagName: String)

### Description

Equivalent to Insert ► XML Tag...  
► Delete...

### Parameter

<i>TagName</i>
----------------

## {XMLTag.Labels}

## Syntax

{XMLTag.Labels *Block*; *Where*}

## PerfectScript Syntax

XMLTag\_Labels (Block: String; Where: Left|Right|Up|Down)

## Description

Equivalent to Insert ▶ XML Tag...  
▶ Labels...

## Parameters

*Block*

*Where*

Left  
Right  
Up  
Down

## {XMLTag.MakeTable}

### Syntax

{XMLTag.MakeTable *Block*}

### PerfectScript Syntax

XMLTag\_MakeTable (Block: String)

### Description

Equivalent to Insert ▶ XML Tag...  
▶ Output...

### Parameter

*Block*

## {XMLTag.Reset}

### Syntax

{XMLTagReset}

### PerfectScript Syntax

XMLTag\_Reset ()

### Description

Equivalent to Insert ▶ XML Tag...  
▶ Delete All...

## **{ZOOM}**

### **Description**

{ZOOM} maximizes and restores the active window.

This command is for compatibility with Quattro Pro for DOS; use [{WindowMaximize}](#) and [{WindowRestore}](#) when developing macros for Quattro Pro for Windows.

To change the zoom factor for a notebook or sheet, use {Notebook.Zoom\_Factor} or {Page.Zoom\_Factor}, respectively.

#### **▶ Note**

- This command is obsolete

#### **▶ Related topics**

# {ZTESTM}

## Syntax

{ZTESTM *InBlock1*, *InBlock2*, *OutBlock*, <*Labels*(0|1)>, <*Alpha*>, <*Difference*>, <*Variance1*>, <*Variance2*>}

## Description

{ZTESTM} performs a two-sample z-Test for means, assuming known variances for each sample. {ZTESTM} is equivalent to the z-Test analysis tool.

## Parameters

<i>InBlock1</i>	One or more numeric cell values representing the first input cells
<i>InBlock2</i>	One or more numeric cell values representing the second input cells
<i>OutBlock</i>	Upper-left cell of the output cells
<i>Labels</i>	1 if labels are located in the first column or row of the input cells; 0 if the input cells do not contain labels; the default is 0
<i>Alpha</i>	Significance level of the test; the default is 0.05
<i>Difference</i>	A value indicating the hypothetical difference in the means between <i>InBlock1</i> and <i>InBlock2</i> ; the default is 0
<i>Variance1</i>	A value indicating the variance of data set one; the default is 0
<i>Variance2</i>	A value indicating the variance of data set two; the default is 0

### ► Note

- This command is obsolete

### ► Related topics

## Numeric Format Codes

<b>Code</b>	<b>Description</b>
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 chart)
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)

A Cross Tab Report before running the AddField macro against it.

	A	B	C	D	E
1	Sum of Sales	Quarter			
2	Year	Q1	Q2	Q3	Q4
3	1991	784052	777343	782978	802026
4	1992	1172853	1150969	1195381	1189067

A Cross Tab Report after the AddField macro has been run.

	A	B	C	D	E
1	Winery	[All] ▼			
2					
3	Sum of Sales	Quarter			
4	Year	Q1	Q2	Q3	Q4
5	1991	784052	777343	782978	802026
6	1992	1172853	1150969	1195381	1189067

A Cross Tab Report before running the CenterLabels macro against it.

	A	B	C	D	E	F
1	Winery	[All] ▼				
2						
3			Quarter			
4	Year	Data	Q1	Q2	Q3	Q4
5	1991	Sum of Cases Sold	5017	4970	5007	5126
6		Sum of Cost Per Case	2840	2840	2840	2840
7	1992	Sum of Cases Sold	7569	7418	7714	7672
8		Sum of Cost Per Case	2840	2840	2840	2840



A Cross Tab Report after the CenterLabels macro has been run.

	A	B	C	D	E	F
1	Winery	[All] ▼				
2						
3			Quarter			
4	Year	Data	Q1	Q2	Q3	Q4
5	1991	Sum of Sales	784052	777343	782978	802026
6		Sum of Cost Per Case	2840	2840	2840	2840
7	1992	Sum of Sales	1172853	1150969	1195381	1189067
8		Sum of Cost Per Case	2840	2840	2840	2840

A Cross Tab Report before running the ColumnSummary macro against it.

	A	B	C	D	E
1	Sum of Sales	Quarter			
2	Year	Q1	Q2	Q3	Q4
3	1991	784052	777343	782978	802026
4	1992	1172853	1150969	1195381	1189067

A Cross Tab Report after the ColumnSummary macro has been run.

	A	B	C	D	E
1	Sum of Sales	Quarter			
2	Year	Q1	Q2	Q3	Q4
3	1991	784052	777343	782978	802026
4	1992	1172853	1150969	1195381	1189067
5	Grand Total	1956905	1928312	1978359	1991093

A partial view of a spreadsheet to be used as the data source for a Cross Tab Report.

	A	B	C	D	E	F	G	H
1	Year	Quarter	Winery	Appellation	Region	Cost Per Case	Cases Sold	Sales
2	1,991	Q1	Duckhorn	Merlot	East	\$165	170	\$28,050
3	1,991	Q2	Beaulieu	Cabernet Sauvignon	North	\$165	170	\$28,050
4	1,991	Q3	Beaulieu	Cabernet Sauvignon	North	\$165	171	\$28,215
5	1,991	Q4	Beaulieu	Cabernet Sauvignon	North	\$165	175	\$28,875

A new Cross Tab Report is created.

	A	B	C	D	E
1	Sum of Sales	Quarter			
2	Year	Q1	Q2	Q3	Q4
3	1991	784052	777343	782978	802026
4	1992	1172853	1150969	1195381	1189067

A Cross Tab Report before running the DataAlignment macro against it.

	A	B	C	D
1			Quarter	
2	Year	Data	Q1	Q2
3	1991	Sum of Sales	784052	777343
4		Sum of Cost Per Case	2840	2840
5	1992	Sum of Sales	1172853	1150969
6		Sum of Cost Per Case	2840	2840

A Cross Tab Report after the DataAlignment macro has been run.

	A	B	C	D	E
1		Quarter	Data		
2		Q1		Q2	
3	Year	Sum of Sales	Sum of Cost Per Case	Sum of Sales	Sum of Cost Per Case
4	1991	784052	2840	777343	2840
5	1992	1172853	2840	1150969	2840

A Cross Tab Report before running the DefineFieldProps macro against it.

	A	B	C	D	E	F
1			Quarter			
2	Year	Data	Q1	Q2	Q3	Q4
3	1991	Sum of Sales	784052	777343	782978	802026
4		Sum of Cases Sold	5017	4970	5007	5126
5	1992	Sum of Sales	1172853	1150969	1195381	1189067
6		Sum of Cases Sold	7569	7418	7714	7672



A Cross Tab Report after the DefineFieldProps macro has specified Sales as the field on which to apply the summary option Max.

	A	B	C	D	E	F
1			Quarter			
2	Year	Data	Q1	Q2	Q3	Q4
3	1991	Sum of Sales	784052	777343	782978	802026
4		Max of Sales	60828	57424	59200	65760
5		Sum of Cases Sold	5017	4970	5007	5126
6	1992	Sum of Sales	1172853	1150969	1195381	1189067
7		Max of Sales	102268	102120	103452	102860
8		Sum of Cases Sold	7569	7418	7714	7672

A Cross Tab Report before running the DisplayInEmptyCell macro against it.

	A	B	C	D	E
1	Sum of Sales	Quarter			
2	Year	Q1	Q2	Q3	Q4
3	1991	118140	119460	121440	122595
4	1992	655740	33000	103452	

A Cross Tab Report after the DisplayInEmptyCell macro has been run.

	A	B	C	D	E
1	Sum of Sales	Quarter			
2	Year	Q1	Q2	Q3	Q4
3	1991	118140	119460	121440	122595
4	1992	655740	33000	103452	TBA

A Cross Tab Report before running the Expand macro against it.

	A	B	C	D	E
1	Winery	[All]			
2	Appellation	[All]			
3					
4	Sum of Sales	Quarter			
5	Year	Q1	Q2	Q3	Q4
6	1991	948032	977104	1005239	1052810
7	1992	965690	978556	996522	1023671

A Cross Tab Report with two new sheets added after the Expand macro has been run

	A	B	C	D	E
1	Winery	Beaulieu			
2	Appellation	[All]			
3					
4	Sum of Sales	Quarter			
5	Year	Q1	Q2	Q3	Q4
6	1991	618140	645140	669575	709450
7	1992	628250	639192	654346	676167

	A	B	C	D	E
1	Winery	Duckhorn			
2	Appellation	[All]			
3					
4	Sum of Sales	Quarter			
5	Year	Q1	Q2	Q3	Q4
6	1991	329892	331964	335664	343360
7	1992	337440	339364	342176	347504

A Cross Tab Report before running the FieldCmp, FieldCmpBase, and FieldCmpltemPreset macros against it.

	A	B	C	D	E
1	Sum of Sales	Quarter			
2	Year	Q1	Q2	Q3	Q4
3	1991	948032	977104	1005239	1052810
4	1992	965690	978556	996522	1023671

A Cross Tab Report after the FieldCmp, FieldCmpBase, and FieldCmpltemPreset macros have been run

	A	B	C	D	E
1	Sum of Sales	Quarter			
2	Year	Q1	Q2	Q3	Q4
3	1991				
4	1992	17658	1452	-8717	-29139

A Cross Tab Report before running the HideField macro against it.

	A	B	C	D	E
1	Sum of Sales	Quarter			
2	Year	Q1	Q2	Q3	Q4
3	1991	948032	977104	1005239	1052810
4	1992	965690	978556	996522	1023671



A Cross Tab Report after the HideField macro has been run.

	A	B	C	D	E
1	Sum of Sales	Quarter			
2	Year	Q1	Q2	Q3	Q4
3	1992	965690	978556	996522	1023671

A Cross Tab Report before running the FieldLabel macro against it.

	A	B	C	D	E
1	Sum of Sales	Quarter			
2	Year	Q1	Q2	Q3	Q4
3	1991	948032	977104	1005239	1052810
4	1992	965690	978556	996522	1023671

A Cross Tab Report after the FieldLabel macro has been run.

	A	B	C	D	E
1	Sum of Sales	Quarter			
2	Years	Q1	Q2	Q3	Q4
3	1991	948032	977104	1005239	1052810
4	1992	965690	978556	996522	1023671

A Cross Tab Report before running the FieldSummary macro against it.

	A	B	C	D	E
1	Sum of Sales	Quarter			
2	Year	Q1	Q2	Q3	Q4
3	1991	948032	977104	1005239	1052810
4	1992	965690	978556	996522	1023671

A Cross Tab Report after the FieldSummary macro has been run.

	A	B	C	D	E	F
1			Quarter			
2	Years	Data	Q1	Q2	Q3	Q4
3	1991	Sum of Sales	948032	977104	1005239	1052810
4		Average of Sales	52668.4444444445	54283.5555555556	55846.6111111111	58489.4444444445
5		Max of Sales	97680	97680	100000	107250
6		Min of Sales	28800	30400	31200	32000
7	1992	Sum of Sales	965690	978556	996522	1023671
8		Average of Sales	53649.4444444445	54364.2222222222	55362.3333333333	56870.6111111111
9		Max of Sales	102300	103125	107250	110550
10		Min of Sales	27200	27360	28480	28525

A Cross Tab Report before running the FormatReport macro against it.

	A	B	C	D	E	F
1			Quarter			
2	Years	Data	Q1	Q2	Q3	Q4
3	1991	Sum of Sales	948032	977104	1005239	1052810
4		Max of Sales	97680	97680	100000	107250
5	1992	Sum of Sales	965690	978556	996522	1023671
6		Max of Sales	102300	103125	107250	110550

A Cross Tab Report after the FormatReport macro has been run.

	A	B	C	D	E	F
1			Quarter			
2	Years	Data	Q1	Q2	Q3	Q4
3	1991	Sum of Sales	948032	977104	1005239	1052810
4		Max of Sales	97680	97680	100000	107250
5	1992	Sum of Sales	965690	978556	996522	1023671
6		Max of Sales	102300	103125	107250	110550

A Cross Tab Report before running the Hide macro against it.

	A	B	C	D	E	F	G	H	I
1	Sum of Sales	Quarter	Winery						
2		Q1		Q2		Q3		Q4	
3	Year	Beaulieu	Duckhorn	Beaulieu	Duckhorn	Beaulieu	Duckhorn	Beaulieu	Duckhorn
4	1991	618140	329892	645140	331964	669575	335664	709450	343360
5	1992	628250	337440	639192	339364	654346	342176	676167	347504



A Cross Tab Report after the Hide macro has been run.

	A	B	C	D	E	F	G	H
1	Sum of Sales	Quarter	Winery					
2		Q1	Q2		Q3		Q4	
3	Year		Beaulieu	Duckhorn	Beaulieu	Duckhorn	Beaulieu	Duckhorn
4	1991	948032	645140	331964	669575	335664	709450	343360
5	1992	965690	639192	339364	654346	342176	676167	347504

A Cross Tab Report before running the LabelEdit macro against it.

	A	B	C	D	E
1	Sum of Sales	Quarter			
2	Year	Q1	Q2	Q3	Q4
3	1991	948032	977104	1005239	1052810
4	1992	965690	978556	996522	1023671

▶ A Cross Tab Report after the LabelEdit macro has been run.

▶ A Cross Tab Report before running the MoveCell macro against it.

A Cross Tab Report after the MoveCell macro has been run.

	A	B	C	D	E	F	G	H	I
1	Sum of Sales								
2		Quarter	Year						
3		Q1		Q2		Q3		Q4	
4		1991	1992	1991	1992	1991	1992	1991	1992
5		948032	965690	977104	978556	1005239	996522	1052810	1023671

▶ A Cross Tab Report before running the MoveField macro against it.

A Cross Tab Report after the MoveField macro has been run.

	A	B	C	D	E	F	G	H	I
1	Sum of Sales								
2		Quarter	Year						
3		Q1		Q2		Q3		Q4	
4		1991	1992	1991	1992	1991	1992	1991	1992
5		948032	965690	977104	978556	1005239	996522	1052810	1023671

A Cross Tab Report before running the PageFilter macro against it.

	A	B	C	D	E
1	Winery	[All] ▼			
2	Appellation	[All] ▼			
3					
4	Sum of Sales	Quarter			
5	Year	Q1	Q2	Q3	Q4
6	1991	948032	977104	1005239	1052810
7	1992	965690	978556	996522	1023671



A Cross Tab Report after the PageFilter macro has been run.

	A	B	C	D	E
1	Winery	Duckhorn ▼			
2	Appellation	[All] ▼			
3					
4	Sum of Sales	Quarter			
5	Year	Q1	Q2	Q3	Q4
6	1991	329892	331964	335664	343360
7	1992	337440	339364	342176	347504

A Cross Tab Report before running the PreserveDataFormat macro against it.

	A	B	C	D	E
1	Winery	[All] ▼			
2					
3	Sum of Sales	Quarter			
4	Year	Q1	Q2	Q3	Q4
5	1991	\$948,032	\$977,104	\$1,005,239	\$1,052,810
6	1992	\$965,690	\$978,556	\$996,522	\$1,023,671

A Cross Tab Report after the PreserveDataFormat macro has been run.

	A	B	C	D	E
1	Winery	[All] ▼			
2					
3	Sum of Sales	Quarter			
4	Year	Q1	Q2	Q3	Q4
5	1991	<b>\$948,032</b>	<b>\$977,104</b>	<b>\$1,005,239</b>	<b>\$1,052,810</b>
6	1992	<b>\$965,690</b>	<b>\$978,556</b>	<b>\$996,522</b>	<b>\$1,023,671</b>

A Cross Tab Report before running the Refresh macro against it.

	A	B	C	D	E
1	Sum of Sales	Quarter			
2	Year	Q1	Q2	Q3	Q4
3	1991	948032	977104	1005239	1052810
4	1992	965690	978556	996522	1023671

A Cross Tab Report after the Refresh macro has been run.

	A	B	C	D	E
1	Sum of Sales	Quarter			
2	Year	Q1	Q2	Q3	Q4
3	1991	948032	977104	1005239	1052810
4	1992	965690	978556	996522	1043671

A Cross Tab Report before running the RowSummary macro against it

	A	B	C	D	E
1	Sum of Sales	Quarter			
2	Year	Q1	Q2	Q3	Q4
3	1991	948032	977104	1005239	1052810
4	1992	965690	978556	996522	1023671

A Cross Tab Report after the RowSummary macro has been run.

	A	B	C	D	E	F
1	Sum of Sales	Quarter				
2	Year	Q1	Q2	Q3	Q4	Grand Total
3	1991	948032	977104	1005239	1052810	3983185
4	1992	965690	978556	996522	1043671	3984439

A Cross Tab Report before running the Show macro against it.

	A	B	C	D	E	F	G	H
1	Sum of Sales	Quarter	Winery					
2		Q1	Q2		Q3		Q4	
3	Year		Beaulieu	Duckhorn	Beaulieu	Duckhorn	Beaulieu	Duckhorn
4	1991	948032	645140	331964	669575	335664	709450	343360
5	1992	965690	639192	339364	654346	342176	676167	347504



A Cross Tab Report after the Show macro has been run.

	A	B	C	D	E	F	G	H	I
1	Sum of Sales	Quarter	Winery						
2		Q1		Q2		Q3		Q4	
3	Year	Beaulieu	Duckhorn	Beaulieu	Duckhorn	Beaulieu	Duckhorn	Beaulieu	Duckhorn
4	1991	618140	329892	645140	331964	669575	335664	709450	343360
5	1992	628250	337440	639192	339364	654346	342176	676167	347504



## **Quattro Pro Macros Help**

Click the Help Topics button to return to the list of topics.



