AbsoluteReference

Syntax

AbsoluteReference()

Description

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

ACTIVATE

Syntax

Activate(*WindowName* As String)

Description

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

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

ADDMENU

Syntax

AddMenu(*MenuPath* As String, *MenuBlock* As 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

Related topics

ADDMENUITEM

Syntax

AddMenuItem(*MenuPath* As String, *ItemName* As String, [*Link* As String], [*Hint* As String], [*HotKey* As String], [*DependString* As String], [*Checked_* As _AddMenuItem_Checked__enum])

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

MenuPath	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).
Name	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 (-).
Link	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 » for details.
Hint	Help text to display in a pop-up window when the command is highlighted (optional)
HotKey	Shortcut key that chooses the command (optional); separate key combinations with a plus sign (+), for example, Alt+F4.
DependString	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.
Checked	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.

Related topics

ADDSERIES

Syntax

AddSeries(*Block* As String, *Name* As 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* As String, *ItemName* As String, [*Link* As String], [*Hint* As String], [*HotKey* As String], [*DependString* As String], [*Checked_* As _AddSubMenuItem_Checked__enum])

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

MenuPath	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).
Name	Name of the command to add; if you want a letter of the name to appear underlined, precede it with an ampersand (&)
Link	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 for details.
Hint	Help text to display on the status line when the command is highlighted (optional)
HotKey	Shortcut key that chooses the command (optional); separate key combinations with a plus sign (+), for example, Alt+F4.
DependString	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.
Checked	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.

Related topics

Alert

Syntax

Alert(*Title_* As String, *Message_* As String, *OKExit_* As String, [*Type_* As Integer], [*Icon_* As Integer], [*DefaultBtn_* As Integer])

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

Title	Title of resultant dialog.
Message	Message text of resultant dialog.
OKExit?	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).
Type	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).
Icon	0 for icon of type Error, 1 for Question, 2 for Warning, 3 for Info (optional; 0 is the default).
DefaultBtn	0 for first button being default, 1 for second, etc... (optional; 0 is the default).

AnalysisExpert

Syntax

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* As String, *OutBlock* As String, [*Grouped* As String], [*Labels_* As _ANOVA1_Labels__enum], [*Alpha* As Double])

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

InBlock	Input cells containing two or more sets of numeric data arranged in columns or rows
OutBlock	Upper left cell of the output cells
Grouped	"C" to group results by column or "R" to group results by row; the default is "C"
Labels	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
Alpha	The significance level at which to evaluate values for the F-statistic; the default is 0.05

Related topics

ANOVA2

Syntax

ANOVA2(*InBlock* As String, *OutBlock* As String, *SampleRows* As Integer, [*Alpha* As Double])

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

InBlock	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
OutBlock	Upper-left cell of the output cells
SampleRows	The number of rows in each sample
Alpha	The significance level at which to evaluate values for the F-statistic; the default is 0.05

Related topics

{ANOVA3}

Syntax

ANOVA3(*InBlock* As String, *OutBlock* As String, [*Labels_* As _ANOVA3_Labels__enum], [*Alpha* As Double])

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

InBlock	Input cells containing two or more sets of numeric data arranged in columns or rows
OutBlock	Upper-left cell of the output cells
Labels	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
Alpha	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.

Parameters

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

 **Note**

- This command is obsolete.

 **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.

Parameters

Location	Cell in which to store the characters read

{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.

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.

Parameters

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

Note

- This command is obsolete.

{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.

0 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

{Appliction.Compatibility<.Option>}

PerfectScript Syntax

Application_Compatibility(<.Option>)

Description

Equivalent to Tools ▸ Settings ▸ Compatibility

Parameters

AlternateMenuBar [String]	Lets you specify which menu to use. 0 "Quattro Pro 8/9" 1 "Quattro Pro 7" 2 "Excel 97" 3 "Custom"
AutoArrayWrap [Boolean]	Lets you specify whether CTRL+SHIFT+ENTER generates an @ARRAY function, or whether Quattro Pro automatically determines whether one is needed. 5 0 CTRL+SHIFT+ENTER generates an @ARRAY function 6 1 Quattro Pro automatically determines whether one is needed
CompatibilityMode [String]	Lets you specify which compatibility default is used. 8 "Quattro Pro 9" 9 "Quattro Pro 8" 10 "Excel 97" 11 "Custom"
Def_Columns_Limit [Numeric]	Lets you specify the maximum number of columns a notebook can contain.
Def_Rows_Limit [Numeric]	Lets you specify the maximum number of rows a notebook can contain.
Def_Sheets_Limit [Numeric]	Lets you specify the maximum number of sheets a notebook can contain.
File_Extension [String]	Lets you specify the default file format.
Min_Number_Sheets [Numeric]	Lets you specify the minimum number of sheets a notebook can contain.
Range_Syntax [String]	Equivalent to Tools ▸ Settings ▸ Compatibility ▸ 3D Syntax.
Sheet_Tab_Label [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.

0 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"}	Lets you specify whether to show or hide portions of the Quattro Pro window, and switches between 3-D syntax schemes.
{Application.Display.Clock_Display Yes|No}	Lets you specify whether to show the Clock Display. This option is obsolete. 1 0 Do not show the Clock Display. 2 1 Show the Clock Display.
{Application.Display.CommentMarkers Yes|No}	Lets you specify whether to show the Comment Markers 4 0 Do not show the Comment Markers. 5 1 Show the Comment Markers.
{Application.Display.Default_View Draft|Page}	Lest you specify whether new Notebooks come up in Draft view or Page Preview view. This option is obsolete.
{Application.Display.Default_Zoom Yes|No}	Lets you specify whether to enable the Default Zoom. This option is obsolete. 8 0 Do not enable the Default Zoom. 9 1 Enable the Default Zoom.
{Application.Display.FormulaMarkers Yes|No}	Lets you specify whether to display the Formula Markers. 11 0 Do not display the Formula Markers. 12 1 Display the Formula Markers.
{Application.Display.History_List Yes|No}	Lets you specify whether to display the File History List off the File menu. 14 0 Do not display the File History List. 15 1 Display the File History List.
{Application.Display.Min_Number_Sheets N}	Lets you specify the default number of sheets on new Notebooks. This option is obsolete.
{Application.Display.Range_Syntax "A..B:A1..B2"|"A:A1..B:B2"}	Lets you switch between 3-D syntax schemes. This option is obsolete.
{Application.Display.RealTime_Prev Yes|No}	Lets you specify whether to enable the RealTime Preview. 19 0 Do not enable the RealTime Preview. 20 1 Enable the RealTime Preview.
{Application.Display.Sheet_Tab_Label Letters|Numbers}	Allows you to choose whether your default sheet tab names are letters (A..IV) or numeric (Sheet1..Sheet256). This option is obsolete.
{Application.Display.Sheet_Tab_Label Letters|Numbers}	Toggles dialog Group Boxes between being 'boxes' or just a line above the group. This option is obsolete.
{Application.Display.Shortcut_Keys Yes|No}	Lets you specify whether to display shortcut keys.

	24 0 Display shortcut keys. 25 1 Do not display shortcut keys.
{Application.Display.Show_GroupBox_As_Line Yes|No}	Toggles dialog Group Boxes between being a 'box', or just a line above the group. This option is obsolete.
{Application.Display.Show_InputLine Yes|No}	Lets you specify whether to show the Input Line. 28 0 Do not show the Input Line 29 1 Show the Input Line.
{Application.Display.Show_PreSelection Yes|No}	Toggles Windows buttons and other controls between being 3-D and flat. This option is obsolete.
{Application.Display.Show_Property_Band Yes|No}	Lets you specify whether to show the Property Bar. This option is obsolete. 32 0 Do not show the Property Bar 33 1 Show the Property Bar
{Application.Display.Show_Scroll_Indicator Yes|No}	Lets you specify whether to show the Scroll Indicators. 35 0 Do not show the Scroll Indicators. 36 1 Show the Scroll Indicators.
{Application.Display.Show_StatusLine Yes|No}	Lets you specify whether to show the the Application Bar. This option is obsolete. 38 0 Do not show the Application Bar. 39 1 Show the Application Bar.
{Application.Display.Show_Tool_Hint Yes|No}	Lets you specify whether to show QuickTips. 41 0 Do not show QuickTips 42 1 Show QuickTips
{Application.Display.Show_Toolbar Yes|No}	Lets you specify whether to show the toolbar. This option is obsolete. 44 0 Do not show the toolbar 45 1 Show the toolbar

Related topics

{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. This option is obsolete.
{Application.File_Options.Startup_Directory String}	Specify a default folder.
{Application.File_Options.TempDir Path}	Specify a folder for QuickTemplates. This option is

	obsolete.
{Application.File_Options.WPDialogs Yes|No}	Use enhanced file dialogs.

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.

0 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 *String*}	Determines the international formats given as options for date display
{Application.International.Language *String*}	Equivalent to Tools ▸ Settings ▸ International ▸ LanguageMode
{Application.International.Language *String*}	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 preceeded 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 *String*}	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. This option is obsolete.
{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.

Example

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

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

1 {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

Note

This command is obsolete.

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

BEEP

Syntax

Beep()

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}
```

Related topics

BLANK

Syntax

Blank(*Blocks* As String)

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

	Location	Cell(s) you want erased

Related topics

BlockCopy

Syntax

BlockCopy(*SourceBlock* As String, *DestBlock* As String, [*ModelCopy_* As _BlockCopy_ModelCopy__enum], [*Formulas_* As _BlockCopy_Formulas__enum], [*Values_* As _BlockCopy_Values__enum], [*Properties_* As _BlockCopy_Properties__enum], [*Objects_* As _BlockCopy_Objects__enum], [*RowCol_*Sizes_ As _BlockCopy_RowCol_Sizes__enum], [*Labels_* As _BlockCopy_Labels__enum], [*Numbers_* As _BlockCopy_Numbers__enum])

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

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

BlockDelete

Syntax

{BlockDelete.*Option*}

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 *Block*, Entire|Partial}	Deletes entire or partial column
{BlockDelete.Pages *Block*, Entire|Partial}	Deletes entire or partial page
{BlockDelete.Rows *Block*, Entire|Partial}	Deletes entire or partial row

BlockFill

Syntax

BlockFill_Block(*Block* As String)

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 *Block*}	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 *Value*}	Sets the first value in the series.
{BlockFill.Step *Value}*	Sets the constant value to add to the Start value or the last value.
{BlockFill.Stop *Value}*	Sets the limit for the fill values.

BlockInsert

Syntax

{BlockInsert.*Option*}

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 *Block,* Entire|Partial}	Inserts complete or partial columns.
{BlockInsert.File *FileName, BeforeBlock*}	Inserts a complete file.
{BlockInsert.Pages *Block,* Entire|Partial}	Inserts complete or partial pages.
{BlockInsert.Rows *Block,* Entire|Partial}	Inserts complete or partial rows.

{BlockMove}

Syntax

BlockMove(*SrcBlock* As String, *DstBlock* As String)

Description

Lets you move a block.

Parameters

SrcBlock	The block you want to move
DstBlock	New location for SrcBlock

Related topics

BlockMovePages

Syntax

BlockMovePages(*SrcPages* As String, BeforePage As String, [*CopyOption_* As _BlockMovePages_CopyOption__enum])}

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.

```
Example:
{BlockMovePages July; August}
```

Parameters

SrcPages	Range of sheets to move
BeforePage	New location for *SrcPages*

Related topics

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 *Block*, *LabelsTop?*(0|1), *LabelsLeft?*(0|1), *LabelsBottom?*(0|1), *LabelsRight?*(0|1), *Intersection?*(0|1)}	Creates cell names from adjacent labels.
{BlockName.Create *BlockName, Block*}	Adds a name for the specified cell to the cell name list.
{BlockName.Delete *BlockName*}	Deletes a selected cell name.
{BlockName.Labels *Block*,Left|Right|Up|Down}	Assigns names to single cells using adjacent labels.
{BlockName.MakeTable *Block*}	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* As String)

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(*SrcBlock* As String, *DstBlock* As String)

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

SourceBlock	Cells to transpose
DestBlock	Cells to hold transposed copy

BlockValues

Syntax

BlockValues(*SrcBlock* As String, *DstBlock* As String)

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

SourceBlock	Cells to copy as values
DestBlock	Cells to hold converted copy

BudgetExpert

Syntax

BudgetExpert()

Description

{BudgetExpert} displays the first Budget Expert dialog box.

CALC

Syntax

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}

Syntax

CapOff()

Description

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

Related topics

ChartExpert

Syntax

ChartExpert()

Description

{ChartExpert} displays the first Chart Expert dialog box.

CLEAR

Syntax

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_* As Integer])

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_* As _ClearContents_PageOnly__enum])

PerfectScript Syntax

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

Description

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

Parameters

PageOnly?	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_* As _ClearFormats_PageOnly__enum])

PerfectScript Syntax

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

Description

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

Parameters

PageOnly?	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

{COLUMNWIDTH}

Syntax

ColumnWidth(*Block* As String, *FirstPane_* As _ColumnWidth_FirstPane__enum, *Mode* As _ColumnWidth_Mode_enum, *Size* As Double)

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

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

Related topics

Comment_Edit

Syntax

Comment_Edit(Value_ As String)

PerfectScript Syntax

Comment_Edit (Value?:String)

Description

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(*Link_* As String, [*Text_* As String], [*Loc_* As String], [*Relative_* As _Comment_EditURL_Relative__enum])

PerfectScript Syntax

Comment_EditURL (Value?:String)

Description

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

ComposeFormula

Syntax

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 cellss 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 *SummaryFunction*}	Specifies the operations to perform on the source cells.
{Consolidate.Go}	Performs the consolidation of the source cells.
{Consolidate.Options *OutputWithFormulas?*(0|1), *LabelsInTopRow?*(0|1), *LabelsInLeftCol?*(0|1)}	Selects options for consolidation.
{Consolidate.Remove *Name*}	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 *Name*}	Saves the current consolidation setup.
{Consolidate.Use *Name*}	Lists saved consolidation setups.

ConsolidateExpert

Syntax

ConsolidateExpert()

Description

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

CONTENTS

Syntax

Contents(*DestCell* As String, *SourceCell* As String, [*Width* As Integer], [*Format* As Integer])

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

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

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

Related topics

CR

Syntax

CR()

Description

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

Related topics

CreateChart

Syntax

CreateChart(*Name* As String)

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, ... *n*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

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

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

Data Option	SUM, AVERAGE, COUNT, % of COLUMN, % of ROW, % of GRAND, or STRING
Row and *Column Options*	SUM, AVERAGE, COUNT, % of COLUMN, % of ROW, % of GRAND, INCREASE, % INCREASE, or STRING

CrossTabReport.AddField

Syntax

CrossTabReport_AddField(*Index_* As Integer, *Type_* As Integer)

PerfectScript Syntax

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

Description

Lets you add a field to the active report.

Parameters

Index	The index of the field
Type	1 Row 2 Column 3 Page 4 Data

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_* As _CrossTabReport_CenterLabels_Enable__enum)

PerfectScript Syntax

CrossTabReport_CenterLabels (Enable?: Boolean)

Description

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

Parameter

Enable	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*_ As _CrossTabReport_ColumnSummary_Enable__enum)

PerfectScript Syntax

CrossTabReport_ColumnSummary (Enable?: Boolean)

Description

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

Parameter

Enable	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_* As Integer)

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

RowOrCol	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(*Props_* As 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

Area	1 Row area 2 Column area 3 Page area 4 Data area
Field Index	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_ As String)

Description

Lets you specify where the report is located.

Parameter

Block	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_* As _CrossTabReport_DisplayInEmptyCell_Enable__enum)

PerfectScript Syntax

CrossTabReport_DisplayInEmptyCell (Enable?: Boolean)

Description

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

Parameter

Enable	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}

{CrossTabReoirt.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_* As String)

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

Name	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

Index1	The field on which you want to base the report expansion.
Index2 [optional]	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_* As Integer)

PerfectScript Syntax

CrossTabReport_FieldCmp (Value?: Numeric)

Description

Lets you specify a comparision option on any given field within a report. Typically, this macro will be used along with the {CrossTabReport_FieldCmpBase}, {CrossTabReport_FieldCmpItem},and {CrossTabReport_FieldCmpItemPreset} macros.

Parameter

Value	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 comparision 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*_ As Integer)

PerfectScript Syntax

CrossTabReport_FieldCmpBase (Value?: Numeric)

Description

Lets you specify the index of the base field.

Parameter

Value	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 comparision. For a more detailed example involving this macro, please see the help for the {CrossTabReport_FieldCmp} macro.

CrossTabReport_FieldCmpItem

Syntax

CrossTabReport_FieldCmpItem(*Value_* As String)

PerfectScript Syntax

CrossTabReport_FieldCmpItemPreset (Value?: String)

Description

Lets you specify the field item to be compared.

Parameter

Value	The index of the field

Example

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

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

CrossTabReport_FieldCmpItemPreset

Syntax

CrossTabReport_FieldCmpItemPreset(*Value_* As Integer)

PerfectScript Syntax

CrossTabReport_FieldCmpItemPreset (Value?: Numeric)

Description

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

Parameter

Value	0 None -1 Previous 1 Next

Example

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

Previous is selected as the type of preset to be used during the comparision. For a more detailed example involving this macro, see the help for the {CrossTabReport_FieldCmp} macro.

CrossTabReport_FieldHide

Syntax

CrossTabReport_FieldHide(*Value*_ As String)

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

Value [semicolon delimited]	Semicolon delimited items

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_* As String)

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

Value	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_* As String)

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

Value [semicolon delimited]	1 Sum 2 Count 3 Average 4 Max 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_* As _CrossTabReport_FormatReport_Enable__enum)

PerfectScript Syntax

CrossTabReport_FormatReport (Enable?: Boolean)

Description

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

Parameter

Enable	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_* As String)

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_* As Integer, *Column_* As Integer)

PerfectScript Syntax

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

Description

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

Parameters

Row	The row you to which you want to move the selected cell.
Column	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_* As Integer, *Source*_Type_ As Integer, *Destination*_Index_ As Integer, *Destination_Type_* As Integer)

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

Source_Area	1 Row area 2 Column area 3 Page area 4 Data area
Source_Index	The numeric index of the source field
Destination_Area	1 Row area 2 Column area 3 Page area 4 Data area
Destination_Index	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_* As String)

PerfectScript Syntax

CrossTabReport_Name (Name?: String)

Description

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

Parameter

Name	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_* As Integer, *Value_* As String)

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

Index	The numeric index of the field to be filtered.
Value	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_* As _CrossTabReport_PreserveDataFormat_Enable__enum)

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

Enable	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_* As _CrossTabReport_ColumnSummary_Enable__enum)

PerfectScript Syntax

CrossTabReport_RowSummary (Enable?: Boolean)

Description

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

Parameter

Enable	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_RowSummary(*Enable_* As _CrossTabReport_RowSummary_Enable__enum)

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_Show()

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_Source(*Block_* As String)

PerfectScript Syntax

CrossTabReport_UpdateDataOnOpen (Enable?: Boolean)

Description

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

Parameter

Enable	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.

DatabaseQuery

Syntax

DatabaseQuery(*Type_* As String, *Name_* As String, QueryString_ As String, Destination_ As String)

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

Type	Type of database to query: "Paradox", "ODBC", or "BDE (Borland Database Engine)".
Name	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.
QueryString	An SQL Statement.
Destination	The destination block of cells where to send the result.

DATE

Syntax

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(*type* As _DbAlias_type_enum, *Path* As String)

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 | PRIV	WORK to specify a Working directory; PRIV to specify a Private directory
Path	path for the Working directory or the Private directory

{DELETEMENU}

Syntax

DeleteMenu(*MenuPath* As String)

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* As String)

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* As String], [*VarName*])

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.

For more details on using {DELVAR} and other OLE automation macro commands, Using OLE Automation Features.

Parameters

VarName	A named variable

{DESCR}

Syntax

DESCR(*InBlock* As String, *OutBlock* As String, [*Grouped* As String], [*Labels_* As _DESCR_Labels__enum], [*Summary_* As _DESCR_Summary__enum], [*Largest* As Integer], [*Smallest* As Integer], [*Confidence* As Double])

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

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

Related topics

{DialogView}

Syntax

DialogView(*Window* As String)

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(*Disable_* As _DialogWindow_Disabled_Disable__enum)

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(*Settings* As String)

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* As String)

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(*Settings* As String)

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; click here  for directions on using the Position Adjust property.

Related topics

{DialogWindow.Title}

Syntax

DialogWindow_Title(*Title* As 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* As 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

{DLL}

Syntax

DLL(*DLLName*_FunctionName As String, [*Argument*])

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

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

Related topics

{DLL.Load}

Syntax

DLL_Load(*DLLName* As String)

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

{DraftViewGoto}

Syntax

DraftViewGoto()

Description

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

Related topics

{EDIT}

Syntax

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}

Syntax

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}

Syntax

EditCopy()

Description

{EditCopy} copies the selected object to the Clipboard.

Related topics

{EditCut}

Syntax

EditCut}

Description

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

Related topics

{EditGoto}

Syntax

EditGoto(*Block* As String, [*Extend_* As _EditGoto_Extend__enum])

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

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

Related topics

{EditPaste}

Syntax

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

{Eval}

Syntax

Eval(*Formula* As String)

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

Formula	The string to evaluate

{EXECAUTO}

Syntax

ExecAuto(*AutoExpr1* As String, [*AutoExpr*])

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.

For more details on using {EXECAUTO} and other OLE automation macro commands, see Using OLE Automation Features.

Parameters

AutoExpr1,2,..	One or more automation expressions

{ExecMacro}

Syntax

ExecMacro(*FileName* As String, *Macro* As String)

PerfectScript Syntax

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

Description

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

Parameters

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

{EXPON}

Syntax

EXPON(*InBlock* As String, *OutBlock* As String, [*Damping* As Double], [*StdErrs_* As _EXPON_StdErrs__enum])

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

InBlock	Input cells containing a single column or row with at least four numeric values; the cells must not contain labels
OutBlock	Upper-left cell of the output cells
Damping	Damping factor used as the exponential smoothing constant; indicates the percentage for error to adjust each prior forecast value; must be ³ 0; the default is 0.3
StdErrs	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* As String, [*GrayScale_* As _ExportGraphic_GrayScale__enum], [*Compression_* As _ExportGraphic_Compression__enum])

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

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

{FileClose} and {FileCloseAll}

Syntax

FileClose([*DoSave_* As _FileClose_DoSave__enum])

FileCloseAll([*DoSave_* As _FileCloseAll_DoSave__enum])

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* As String, *Blocks* As String, *Operation* As _FileCombine_Operation_enum)

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

Filename	Name of the file to combine
Blocks	Selection or selections within Filename to combine (optional)

{FileExit}

Syntax

FileExit([*DoSave_* As _FileExit_DoSave__enum])

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

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

 **Related topics**

FileExtract

Syntax

FileExtract(*What* As _FileExtract_What_enum, *Blocks* As String, *FileName* As String, [*Option* As _FileExtract_Option_enum])

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

Blocks	Selection or selections to extract
Filename	Name of the new file containing Blocks

FileImport

Syntax

FileImport(*FileName* As String, *Method* As String)

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* As String])

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

TemplateName	The name of a QuickTemplate

{FileOpen}

Syntax

FileOpen(*FileName* As String, [*Option* As _FileOpen_Option_enum])

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

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

{FileRetrieve}

Syntax

FileRetrieve(*FileName* As String, [*Option* As _FileRetrieve_Option_enum])

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

Filenam	Name of the file to retrieve

e

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

Syntax

FileSave([*Option* As _FileSave_Option_enum])

FileSaveAll([*Mode* As _FileSaveAll_Mode_enum])

FileSaveAs(*FileName* As String, [*Option* As _FileSaveAs_Option_enum], [*reserved* As Integer], [*FileType* As String])

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 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|	Saves the file over a previous version with the same name

Backup|Confirm>}	
{FileSaveAs *Filename,* <Replace | Backup | Confirm>,, <*FileType*>}	Saves the notebook under a new name you specify

Related topics

{FileSend}

Syntax

FileSend([*FileName* As String])

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

{FileSend <*Filename*>}	Sends selected text or an entire notebook by e-mail

{ FileVersion_Retrieve }

Syntax

FileVersion_Retrieve(*Filename_* As String)

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* As String, *xoffset* As Double, *yoffset* As Double)

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

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

Related topics

{FLOATCREATE}

Syntax

FloatCreate(*type* As String, *UpperCell* As String, *xoffset* As Double, *yoffset* As Double, *LowerCell* As String, *xoffset2* As Double, *yoffset2* As Double, [*TextOrStartCorner*])

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..Dll . 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

Type	Floating object to create: Chart, Button, Line, Arrow, Rect, Rounded_Rect, Ellipse, or Text
UpperCell	Cell containing the upper-left corner of the chart or macro button
xoffset	Offset in twips from the left edge of *UpperCell* to the left edge of the floating object
yoffset	Offset in twips from the top edge of *UpperCell* to the top edge of the floating object
LowerCell	Cell containing the lower-right corner of the chart or macro button
xoffset2	Offset in twips from the left edge of *LowerCell* to the right Edge of the floating object
yoffset2	Offset in twips from the top edge of *LowerCell* to the bottom edge of the floating object
Text	For Chart, the named chart to display; for Button, the button text
StartCorne	For Line or Arrow, a number representing the starting corner; 1 =

r	northwest, 2 = northeast, 3 = southeast, 4 = southwest (for example, an arrow pointing up and to the right would have a *StartCorner* of 4)

Related topics

{FLOATMOVE}

Syntax

FloatMove(*UpperCell* As String, *xoffset* As Double, *yoffset* As Double)

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

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

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* As String, *xoffset* As Double, *yoffset* As Double, *LowerCell* As String, *xoffset2* As Double, *yoffset2* As Double)

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

UpperCell	Cell containing the new upper-left corner of the chart or macro button
xoffset	Offset in twips from the left edge of UpperCell to the left edge of the floating object
yoffset	Offset in twips from the top edge of UpperCell to the top edge of the floating object
LowerCell	Cell containing the new lower-right corner of the chart or macro button
xoffset2	Offset in twips from the left edge of LowerCell to the right edge of the floating object
yoffset2	Offset in twips from the top edge of LowerCell to the bottom edge of the floating object

Related topics

{FLOATTEXT}

Syntax

FloatText(*String* As 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

{Form}

Syntax

Form([*Block*_ As String])

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* As String, *OutBlock* As String, [*Inverse_* As _FOURIER_Inverse__enum])

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

InBlock	One or more numeric cell values representing the input cells; can be real or complex numbers; the number of values in InBlock must be a power of 2 between 2 and 1024 inclusive (for example, 2, 4, 8, 16,...); if the number of values in InBlock does not equal a power of 2, pad the cells with additional zeros
OutBlock	Upper-left cell of the output cells
Inverse	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 *Block*}	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 *Block*}	Specifies the cells or list of cells containing values to be counted

{FTESTV}

Syntax

FTESTV(*InBlock1* As String, *InBlock2* As String, *OutBlock* As String, [*Labels_* As _FTESTV_Labels__enum])

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

InBlock1	The first input cells containing a column or row of numeric values
InBlock2	The second input cells containing a column or row of numeric values
OutBlock	Upper-left cell of the output cells
Labels	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}

Syntax

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

{GetCellFormula}

Syntax

GetCellFormula(*Cell* As String) As String

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* As String) As String

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* As String, [*Path* As String])

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

Block	Cells to enter list of files into
Path	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

{GetObjectPageContents}

Syntax

GetObjectPageContents(*Block_* As String, [*Object_* As String])

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

Block	Cells in which to store object names
ObjectType	All (default), Dialog, Chart, Map, and SlideShow.

Tip

- 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* As String, *ObjectProperty* As String)

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

Cell	Cell in which to store the property setting
Object	Name of the object to study
Property	Property of the object to study

Related topics

{GETPROPERTY}

Syntax

GetProperty(*Cell* As String, *PropertyName* As String)

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 (see Property Reference for a list of properties); its setting is stored in *Cell*.

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* As String)

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

{GraphCopy}

Syntax

GraphCopy(*FromGraph* As String, *DestGraph* As String, [*Style_* As _GraphCopy_Style__enum], [*Data_* As _GraphCopy_Data__enum], [*Annotations_* As _GraphCopy_Annotations__enum])

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

FromChart	Chart containing the style, data, or annotation objects to copy
DestChart	New chart (the copy)
Style?	Whether to copy properties that affect the appearance of the chart: yes (1), no (0)
Data?	Whether to copy chart data: yes (1), no (0)
Annotations?	Whether to copy annotation objects: yes (1), no (0)

{GraphDeactivate}

Syntax

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* As String)

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

Name	Name of the chart to delete

{GraphEdit}

Syntax

GraphEdit(*Name* As String, [*InPlace_* As _GraphEdit_InPlace__enum])

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

Name	Name of the chart to edit
InPlace?	Whether to edit the chart in place on a notebook sheet; 0 = no, 1 = yes; the default is 0

{GraphGallery}

Syntax

GraphGallery(*GraphStyle* As String, *ColorScheme* As String)

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
- Default
- Grayscale
- Pastels
- Fire and Ice
- 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

ChartStyle	The style of chart; see {GraphSettings.Type} for a list of chart types
ColorScheme	The color scheme used for the chart

{GraphNew}

Syntax

GraphNew(*Name* As String, [*UseCurrentBlock_* As _GraphNew_UseCurrentBlock__enum])

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

Name	Name of the new chart
UseCurrentBlock?	Whether to chart the current selected cells; 0 = no, 1 = yes; the default is 0name of the new chart

{GraphSettings_Check}

Syntax

GraphSettings_Check()

PerfectScript Syntax

GraphSettings_Check ()

Description

Example

{GraphSettings_Reset}

Syntax

GraphSettings_Reset()

PerfectScript Syntax

GraphSettings_Reset ()

Description

Example

{GraphSettings.Titles}

Syntax

GraphSettings_Titles(*Main* As String, *Sub* As String, *XAxis* As String, *YAxis* As String, *Y2Axis* As String)

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

Main	Main title of the chart
Sub	Title appearing below the main title of the chart
X-Axis	Title of the chart's x axis
Y-Axis	Title of the chart's y axis
Y2-Axis	Title of the chart's secondary y axis

Related topics

{GraphSettings_Type}

Syntax

GraphSettings_Type(*type* As String)

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(*GraphName* As String, [*MoreGraphName*])

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

ChartName 1	Name of the first chart to display (optional)
ChartName 2	Name of the second chart to display (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 *GroupName*, *StartPage*, *EndPage*}	Creates sheet groups
{Group.Delete *GroupName*}	Deletes the currently selected sheet group
{Group.ResetNames}	Clears all group names in the notebook

{GroupObjects}

Syntax

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}

Syntax

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.

Note

- This command is obsolete.

Related topics

{HISTOGRAM}

Syntax

HISTOGRAM(*InBlock* As String, *OutBlock* As String, [*BinBlock* As String], [*Pareto_* As _HISTOGRAM_Pareto__enum], [*Cum_* As _HISTOGRAM_Cum__enum])

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

InBlock	Input cells containing one or more columns or rows of numeric values; the cells must not contain labels
OutBlock	Upper-left cell of the output cells
BinBlock	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
Pareto	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
Cum	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* As Integer)

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.

Example

{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

{HPAGE}

Syntax

HPage(*Distance* As Integer)

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.

Parameters

Distance	Distance in screens to scroll the active notebook horizontally

Related topics

{IMFORMAT}

Syntax

IMFORMAT(*Format* As Integer)

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

Format	Flag indicating what suffix and format to use for imaginary coefficient of complex number; the default is 1; 1 = *x* + *y*i, 2 = *x* + *y*j, 3 = *x* + i*y*, 4 = *x* + j*y*

{ImportGraphic}

Syntax

ImportGraphic(*FileName* As String)

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

Filename	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

{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.

{INDICATE}

Syntax

Indicate([*String* As 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

{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(ObjectTypeOrFilename As String, [DisplayAsIcon_ As _InsertObject_DisplayAsIcon__enum], [Linked_ As _InsertObject_Linked__enum])

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

Filename	File that you want to link/embed as an object
DisplayAsIcon	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
Linked?	Whether to link to the file; 0 to not link; 1 to link; the default is 0

Parameters 2

ObjectType	Type of object to insert (the name of an OLE server)
DisplayAsIcon	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 ()

Description

Example

{InsertObject_TextArt}

Syntax

InsertObject_TextArt()

PerfectScript Syntax

InsertObject_TextArt ()

Description

Example

{InsertPageBreak}

Syntax

{InsertPageBreak.*Option*}

PerfectScript Syntax

InsertPageBreak_Create(*Row* As Integer, *Column* As Integer)

InsertPageBreak_Delete(*Row* As Integer, *Column* As Integer)

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}

Syntax

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 As String)

Invert_Go()

Invert_Source(Block As 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 *Block*}	Specifies the upper-left cell of the area where you want to write the inverted matrix
{Invert.Go}	Inverts the selected matrix
{Invert.Source *Block*}	Specifies the matrix you want to invert

Related topics

{IsAutoObj}

Syntax

IsAutoObj(*Object* As String)

PerfectScript Syntax

IsAutoObj (Object: String)

Description

Parameter

Object

{LET}

Syntax

Let(*Cell* As String, *Value*)

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 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

Location	Cell in which to store the specified value
Value	Numeric or string value to be stored in *Location*
Type	String or value; string (or s) stores the value or formula as a label, and value (or v) stores the actual value or value resulting from a formula (optional)

Related topics

{Links}

Syntax

{Links.*Option*}

PerfectScript Syntax

Links_Change(*OldName* As String, *NewName* As String)

Links_Delete(*LinkName* As String)

Links_Open(*LinkName* As String)

Links_Refresh(*LinkName* As 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 *OldName*, *NewName*}	Switches links from one file to another
{Links.Delete *LinkName*|*} (* = all links)	Deletes notebook links
{Links.Open *LinkName*|*} (* = all links)	Opens files linked to the active notebook
{Links.Refresh *LinkName*|*} (* = all links)	Refreshes links to unopened files

{MACROS}

Syntax

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}

Syntax

MapExpert()

Description

{MapExpert} displays the first Map Expert dialog box. The macro has no arguments.

{MCORREL}

Syntax

MCORREL(*InBlock* As String, *OutBlock* As String, [*Grouped* As String], [*Labels_* As _MCORREL_Labels__enum])

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

InBlock	Input cells containing two or more sets of numeric data arranged in columns or rows
OutBlock	Upper-left cell of the output cells
Grouped	"C" to group results by column or "R" to group results by row; the default is "C"
Labels	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* As String, *OutBlock* As String, [*Grouped* As String], [*Labels_* As _MCOVAR_Labels__enum])

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

InBlock	Input cells containing two or more sets of numeric data arranged in columns or rows
OutBlock	upper-left cell of the output cells
Grouped	"C" to group results by column or "R" to group results by row; the default is "C"
Labels	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

{MOVEAVG}

Syntax

MOVEAVG(*InBlock* As String, OutB*l*ock As String, *Interval* As Integer, [*StdErrs_* As _MOVEAVG_StdErrs__enum])

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

InBlock	Input cells containing a single column or row with at least four numeric values; the cells must not contain labels
OutBlock	Upper-left cell of the output cells
Interval	Number of values to include in the moving average; the default is 3
StdErrs	Flag indicating whether to include standard error values in the OutBlock: yes (1) or no (0); the default is 0

Related topics

{MOVETO}

Syntax

MoveTo(*x* As Double, *y* As Double)

PerfectScript Syntax

MoveTo (x:Numeric; y:Numeric)

Description

{MOVETO} moves all selected objects in the active window (dialog, chart, or Objects sheetg 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* As String], [*Rate* As Double], [*Term* As Double], [*OrigBal* As Double], [*EndBal* As Double], [*LastYear* As Double])

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

OutBlock	Upper-left cell of the output cells
Rate	Yearly interest rate; the default is 0.12
Term	Number of years in the loan; the default is 30 years; can be a fractional value to designate months (for example, 3+5/12)
OrigBal	Original loan balance; the default is $100,000
EndBal	Balance at loan completion; the default is $0
LastYear	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* As String, [*CurrBal* As Double], [*CurrRate* As Double], [*RemTerm* As Double], [*CandPctFees* As Double], [*CandRate* As Double])

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

OutBlock	Upper-left cell of the output cells
CurrBal	Remaining principal on the current loan
CurrRate	Annual interest rate on the current loan
RemTerm	Remaining term on the current loan
CandPctFees	Percentage fees ("points") for the candidate loan
CandRate	Annual interest rate for the candidate loan

Related topics

{Multiply}

Syntax

{Multiply.*Option*}

PerfectScript Syntax

Multiply_Destination(Block As String)

Multiply_Go()

Multiply_Matrix_1(Block As String)

Multiply_Matrix_2(Block As 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}

Syntax

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* As String)

NamedStyle_Define(*StyleName* As String, Align_ As _NamedStyle_Define_Align__enum, *NumericFormat_* As _NamedStyle_Define_NumericFormat__enum, *Protection_* As _NamedStyle_Define_Protection__enum, *Lines_* As _NamedStyle_Define_Lines__enum, *Shading_* As _NamedStyle_Define_Shading__enum, *Font_* As _NamedStyle_Define_Font__enum, *TextColor_* As _NamedStyle_Define_TextColor__enum)

NamedStyle_Delete(*StyleName* As String)

NamedStyle_Font(*Settings* As String)

NamedStyle_Line_Drawing(*Settings* As String)

NamedStyle_Numeric_Format(*Settings* As String)

NamedStyle_Protection(*Settings* As String)

NamedStyle_Shading(*Settings* As String)

NamedStyle_Text_Color(*ColorID* As Integer)

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,0,0,0,0,0,1,1}
```

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()

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

CellAtPointer?	Specifies which cell should be active when the pane switches (0 or 1, optional)

Related topics

{NEXTTOPWIN}

Syntax

NextTopWin()

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()

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_Display}

Syntax

Notebook_Display(*Settings* As String)

PerfectScript Syntax

Notebook_Display_Objects(Mode As String)

Notebook_Display_Show_HorizontalScroller(Show_ As _Notebook_Display_Show_HorizontalScroller_Show__enum)

Notebook_Display_Show_HorizontalScroller(Show_ As _Notebook_Display_Show_HorizontalScroller_Show__enum)

Notebook_Display_Show_Tabs(Show_ As _Notebook_Display_Show_Tabs_Show__enum)Notebook_Display_Show_Tabs(Show_ As _Notebook_Display_Show_Tabs_Show__enum)

Notebook_Display_Show_VerticalScroller(*Show_* As _Notebook_Display_Show_VerticalScroller_Show__enum)

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(*Mode* As String)

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(*Enable*_ As _Notebook_Macro_Library_Enable__enum)

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_Password}

Syntax

Notebook_Password(*Password* As String)

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.

Tips

- Before specifying a password, set the password level using {Notebook.Password_Level}.

Related topics

{Notebook_Password_Level}

Syntax

Notebook_Password_Level(*Settings_* As String)

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(*Settings* As String)

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 *Title* }	Specifies a title for the notebook
{Notebook.Summary.Subject *Subject*}	Specifies a subject for the notebook
{Notebook.Summary.Author *Author*}	Specifies an author for the notebook
{Notebook.Summary.Keywords *Keywords*}	Specifies keywords for the notebook
{Notebook.Summary.Comments *Comments*}	Specifies comments for the notebook

{Notebook_System}

Syntax

Notebook_System(*Enable*_ As _Notebook_System_Enable__enum)

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(*Factor* As Integer)

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}

Syntax

NumOff()

Description

{NUMOFF} and {NUMON} are equivalent to Num Lock off and Num Lock on, respectively.

Related topics

ObjectsPageGoto()

Syntax

ObjectsPageGoto()

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 differenty 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

{OnlineService}

Syntax

OnlineService(*ServiceName* As String, [*Arguments* As String])

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

ServiceName	A string indicating the type of online service to use.
Arguments	A string indicating the command line to pass to the service.

{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 *Constraint#, Cell,* <=|>=|=|Integer, *Constant*}	Adds a new constraint
{Optimizer.Answer_Reporting *Cell*}	Specifies the cells for the Answer Report
{Optimizer.Auto-scale 0|1}	Automatically scales variables to achieve a target value
{Optimizer.Change *Constraint#, Cell,* <=|>=|=| Integer, *Constant*}	Edits the selected constraint
{Optimizer.Delete *Constraint#*}	Removes the selected constraint
{Optimizer.Derivatives Central|Forward}	Selects differencing for estimates of partial derivatives
{Optimizer.Detail_Reporting *Cell*}	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 *Value*}	Sets the maximum number of iterations or trails
{Optimizer.Max_Time *Value*}	Indicates how long Optimizer can spend looking for the best solution
{Optimizer.Model_Cell *Cell*}	Saves cells of Optimizer settings for later use

{Optimizer.Precision *Value*}	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 *SolutionCell*}	Specifies the cell whose value you want Optimizer to measure
{Optimizer.Solution_Goal Max|Min|None|Target Value}	Specifies maximum, minimum, and target values
{Optimizer.Solve}	Finds a solution to the defined problem
{Optimizer.Target_Value *Value*}	Specifies the value to be reached by the formula in the Solution Cell
{Optimizer.Tolerance *Value*}	Indicates the maximum percentage a solution can differ from a theoretical optimum integer solution
{Optimizer.Variable_Cells *Cell(s)*}	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:

Property	Description
Conditional_Color	Changes the color of specific types of data in the active sheet: values above or below a specified range, and ERR values
Default_Width	Sets the default width of all columns in the active sheet
Display	Sets display characteristics for the active sheet
Name	Controls the name of the active sheet
Protection	Turns on protection in the active sheet
Tab_Color	Changes the tab color of the active sheet
Zoom_Factor	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 thesheet 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* As Integer)

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"}
```

Options

{Page.Display *DisplayZeros?*(Yes|No), *RowBorders?*(Yes|No), *ColumnBorders?*(Yes|No), *HorzGridLines?*(Yes|No), *VertGridLines?*(Yes|No)}	Sets display characteristics for the active sheet
{Page.Display.Borders "*RowBorders?*(Yes|No), *ColumnBorders?*(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 "*HorizGridLines?*(Yes|No), *VertGridLines?*(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.Objects 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}.

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

Apply	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

Value	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

Enable	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_* As _ParseExpert_ColumnWidths_Apply__enum)

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_ As _ParseExpert_ConsecutiveAsOne_Apply__enum)

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_* As String)

PerfectScript Syntax

ParseExpert_DataType (Type?:String)

Description

Lets you specify which additional parse options are displayed.

Parameter

Type	"Fixed" Display the fixed parse options. "Delmited" Display the delimited parse options.

{ParseExpert.DelimiterType}

Syntax

ParseExpert_DelimiterType(*Type_* As String)

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_* As _ParseExpert_IgnoreNonConformingRows_Apply__enum)

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

Apply	0 Do not skip the lines. 1 Skip the lines.

{ParseExpert_InputBlock}

Syntax

ParseExpert_InputBlock(*Block_* As String)

PerfectScript Syntax

ParseExpert_InputBlock (Block?:String)

Description

Lets you specify the range of cells to parse.

Parameter

Block	The range of cells

ParseExpert_InputFile

Syntax

ParseExpert_InputFile(*Filename_* As String)

PerfectScript Syntax

ParseExpert_InputFile (Filename?:String)

Description

Lets you specify the name of the file.

Parameter

Filename	The name of the file

{ParseExpert.InputType}

Syntax

ParseExpert_InputType(*Type_* As String)

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_* As _ParseExpert_JoinBrokenLines_Apply__enum)

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_ As Integer)

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_* As String)

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_* As String)

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_* As Integer)

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_* As _ParseExpert_PageLengthEnabled_Apply__enum)

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

{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_* As String)

PerfectScript Syntax

ParseExpert_RowDelimiterString (Value?:String)

Description

Lets you specify the row delimiter

Parameter

Value	The row delimiter

{ParseExpert_RowDelimiterTypeComma}

Syntax

ParseExpert_RowDelimiterTypeComma(*Enable_* As _ParseExpert_RowDelimiterTypeComma_Enable__enum)

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_* As _ParseExpert_RowDelimiterTypeMultiSpace_Enable__enum)

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_* As _ParseExpert_RowDelimiterTypeOther_Enable__enum)

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_* As _ParseExpert_RowDelimiterTypeReturn_Enable__enum)

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_* As _ParseExpert_RowDelimiterTypeSemiColon_Enable__enum)

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_* As _ParseExpert_RowDelimiterTypeSpace_Enable__enum)

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*_ As _ParseExpert_RowDelimiterTypeTab_Enable__enum)

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_* As String)

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_* As String)

PerfectScript Syntax

ParseExpert_SheetDelimiterString (Value?: String)

Description

Lets you specify the sheet delimiter.

Parameter

Value	The sheet delimiter

{ParseExpert_SheetDelimiterTypeComma}

Syntax

ParseExpert_SheetDelimiterTypeComma(*Enable_* As _ParseExpert_SheetDelimiterTypeComma_Enable__enum)

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_* As _ParseExpert_SheetDelimiterTypeMultiSpace_Enable__enum)

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_* As _ParseExpert_SheetDelimiterTypeOther_Enable__enum)

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_* As _ParseExpert_SheetDelimiterTypeReturn_Enable__enum)

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_* As _ParseExpert_SheetDelimiterTypeSemiColon_Enable__enum)

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_* As _ParseExpert_SheetDelimiterTypeSpace_Enable__enum)

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_* As _ParseExpert_SheetDelimiterTypeTab_Enable__enum)

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_* As _ParseExpert_Skip1stChar_Apply__enum)

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_* As String)

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_* As String)

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"

{PasteFormat}

Syntax

PasteFormat(*LinkType* As String)

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}

Syntax

PasteLink()

Description

{PasteLink} sets up a DDE link to another application.

Related topics

PasteSpecial

Syntax

PasteSpecial([*Properties* As String], [*FormulaCells* As String], [*LabelCells* As String], [*NumberCells* As String], [*FormulaValues* As String], [*Transpose* As String], [*NoBlanks* As String])

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

{POKE}

Syntax

Poke(*DDEChannel* As Integer, *Destination* As String, *DataToSend* As String)

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.

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
```

Parameters

DDEChannel	Channel ID number of the application to send information to
Destination	Location in the application that receives the information being sent
DataToSend	Cells containing the information to send to the application

Related topics

{Preview}

Syntax

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 *NamedSetting*}	Creates a named print setting using the name in the New Set text box Replaces the settings stored under the selected name with the current print settings
{Print.Delete *NamedSetting*}	Deletes the selected named setting
{Print.Use *NamedSetting*}	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 *FooterString*}	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 *HeaderString*}	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 *Block*}	Prints the cells you specify
{Print.Copies *Value*}	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 *Value*}	Specifies the beginning and ending pages in the document to print
{Print.PrinterSetup *Printer; Port; PrintToFile* (0|1)*; Filename; CancelOverwrite* (0) | *Replace* (1) | *Backup* (2) | *Append* (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 *Block*}	Adds the cell entries you specify as headings to print at the left of each printed page
{Print.Lines_Between_Blocks *Value*}	Specifies how many blank lines to print between each group of cells
{Print.Lines_Between_Pages *Value*}	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 *Block*}	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* As String, *InBlock2* As String, *OutBlock* As String, [*Labels_* As _PTTESTM_Labels__enum], [*Alpha* As Double], [*Difference* As Double])

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

InBlock1	The first input cells containing a column or row of numeric values
InBlock2	The second input cells containing a column or row of numeric values
OutBlock	Upper-left cell of the output cells
Labels	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
Alpha	Significance level of the test; the default is 0.05
Difference	Hypothetical mean difference; the default is 0

Related topics

{PTTESTV}

Syntax

PTTESTV(*InBlock1* As String, *InBlock2* As String, *OutBlock* As String, [*Labels_* As _PTTESTV_Labels__enum], [*Alpha* As Double])

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 indepependent (rather than paired) samples with unequal variances. {PTTESTV} is equivalent to the t-Test analysis tool.

Parameters

InBlock1	The first input cells containing a column or row of numeric values
InBlock2	The second input cells containing a column or row of numeric values
OutBlock	Upper-left cell of the output cells
Labels	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
Alpha	Significance level of the test; the default is 0.05

Related topics

{PUT}

Syntax

Put(*Block* As String, *Column* As Integer, *Row* As Integer, Value)

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.

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

Location	Cells within which Value will be stored, either as a value or label, as specified by Type
Column#	Number of columns into the specified cells to store Value
Row#	Number of rows into the specified cells to store Value
Value	String or numeric value
Type	String or value; string (or s) stores the value or formula as a label, and value (or v) stores the actual value or value resulting from a formula (optional)

Related topics

{PUTBLOCK}

Syntax

PutBlock(*Data*, [*Block* As String], [*Date*_ As _PutBlock_Date__enum])

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

Data	Entry to type
Block	Cells to type *Data* in (optional)
Date?	Whether to enter *Data* as a date (1) or a label (0)

Related topics

{PUTBLOCK2}

Syntax

PutBlock2(*Data*, [*Block* As String])

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

Data	Entry to type
Block	Cells to type *Data* in (optional)

Related topics

{PUTCELL}

Syntax

PutCell(*Data*, [*Date_* As _PutCell_Date__enum])

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

Data	String to type into the active cell
Date?	Whether to enter *Data* 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

{QUERY}

Syntax

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 *Block*}	Specifies cells containing search conditions, including field names
{Query.Database_Block *Block*}	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 *Block*}	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(*Enable_* As _QuickCorrect_Enable__enum)

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([*Block_* As String], [*OpCode1_* As String], [*Value1_* As String], [*Conditional1_* As String], [*OpCode2_* As String], [*Value2_* As String], [*Conditional2_* As String], [*OpCode3_* As String], [*Value3_* As String])

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

OpCode#	"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."
Arg#	Can be numeric, or a string. Wildcards are not valid.
Conditional#	AND or OR

Related topics

{QuickFilter.Toggle}

Syntax

QuickFilter_Toggle([*Block_* As String])

PerfectScript Syntax

QuickFilter_Toggle ([Block?:String])

Description

Turns on/off QuickFilters for the current cells.

Related topics

{QuickFilter.TopGo}

Syntax

QuickFilter_TopGo([*Block_* As String], [*OpCode1_* As String], [*Value1_* As Integer])

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

OpCode	"Top value," "top percent," "bottom value," "bottom percent"
Arg	Must be numeric. Wildcards are not valid.

Related topics

{QuickFunction}

Syntax

QuickFunction(Name_ As String, [Block_ As String])

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

Name	SUM, MIN, MAX, AVG, PUREAVG, MULT, PMT, RATE, IRATE, TERM, PV, FV
Block	A database block including field labels and records

Related topics

{QUIT}

Syntax

Quit()

Description

{QUIT} ends all macro execution, and returns control of Quattro Pro to you.

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}

Syntax

RANDOM(*OutBlock* As String, *Columns* As Integer, *Rows* As Integer, type As _RANDOM_Type_enum, *Seed* As Double, Parameter1, [*Parameter2* As Double], [*Parameter3* As Double], [*Parameter4* As Double], [*Parameter5* As Double]

Parameters

OutBlock	Upper-left cell of the output cells
Columns	A value indicating the number of random-number sets to generate; default is the number of columns in *OutBlock*
Rows	A value indicating the number of rows of random numbers to generate for each column
6	Indicates patterned distribution
Seed	Starting number for the random-number-generation algorithm
LowerBound	A value indicating the lower bound on the set of numbers to generate
UpperBound	A value indicating the upper bound on the set of numbers to generate
Step	Increment value between *LowerBound* and *UpperBound*
RepeatNumber	A value indicating the number of times to repeat each value
RepeatSequence	A value indicating the number of times to repeat each sequence of values

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

OutBlock	Upper-left cell of the output cells
Columns	A value indicating the number of random-number sets to generate; default is the number of columns in *OutBlock*
Rows	A value indicating the number of rows of random numbers to generate for each column
1	Indicates uniform distribution
Seed	Starting number for the random-number-generation algorithm
LowerBound	A value indicating the lower bound on the set of numbers to generate
UpperBound	A value indicating the upper bound on the set of numbers to generate

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

OutBlock	Upper-left cell of the output cells
Columns	A value indicating the number of random-number sets to generate; default is the number of columns in *OutBlock*
Rows	A value indicating the number of rows of random numbers to generate for each column
2	Indicates normal distribution
Seed	Starting number for the random-number-generation algorithm
Mean	A value indicating the mean of the set of numbers to generate
SDev	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

OutBlock	Upper-left cell of the output cells
Columns	A value indicating the number of random-number sets to generate; default is the number of columns in *OutBlock*
Rows	A value indicating the number of rows of random numbers to generate for each column
3	Indicates Bernoulli distribution
Prob	Starting number for the random-number-generation algorithm
Seed	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

OutBlock	Upper-left cell of the output cells
Columns	A value indicating the number of random-number sets to generate; default is the number of columns in *OutBlock*
Rows	A value indicating the number of rows of random numbers to generate for each column
4	Indicates binomial distribution
Seed	Starting number for the random-number-generation algorithm
Prob	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
Trials	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

OutBlock	Upper-left cell of the output cells
Columns	A value indicating the number of random-number sets to generate; default is the number of columns in *OutBlock*
Rows	A value indicating the number of rows of random numbers to generate for each column
5	Indicates Poisson distribution
Seed	Starting number for the random-number-generation algorithm
Lambda	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

OutBlock	Upper-left cell of the output cells
Columns	A value indicating the number of random-number sets to generate; default is the number of columns in *OutBlock*
Rows	A value indicating the number of rows of random numbers to generate for each column
6	Indicates patterned distribution
Seed	Starting number for the random-number-generation algorithm
LowerBound	A value indicating the lower bound on the set of numbers to generate
UpperBound	A value indicating the upper bound on the set of numbers to generate
Step	Increment value between *LowerBound* and *UpperBound*
RepeatNumber	A value indicating the number of times to repeat each value
RepeatSequence	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

OutBlock	Upper-left cell of the output cells
Columns	A value indicating the number of random-number sets to generate; default is the number of columns in *OutBlock*
Rows	A value indicating the number of rows of random numbers to generate for each column
7	Indicates discrete distribution
Seed	Starting number for the random-number-generation algorithm
InBlock	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* As String, *OutBlock* As String, [*Grouped* As String], [*Labels_* As _RANKPERC_Labels__enum])

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

InBlock	Input cells containing one or more columns or rows of numeric values
OutBlock	Upper-left cell of the output cells
Grouped	"C" to group results by column or "R" to group results by row; the default is "C"
Labels	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

{RECALC}

Syntax

Recalc(*Block* As String, [*Condition*], [*Iteration* As Integer])

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

Location	Cells to recalculate
Condition	Condition to be met before recalculation is halted (optional)
Iteration #	Maximum number of times to recalculate *Location* trying to meet *Condition* (optional)

Related topics

{RECALCCOL}

Syntax

RecalcCol(*Block* As String, [*Condition*], [*Iteration* As Integer])

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

Location	Cells to recalculate
Condition	Condition to be met before recalculation is halted (optional)
Iteration	Maximum number of times to recalculate *Location* trying to meet *Condition* (optional)
#	

Related topics

{RefreshMenuBar}

Syntax

RefreshMenuBar()

PerfectScript Syntax

RefreshMenuBar ()

Description

Refreshes the menu bar.

{REGRESS}

Syntax

REGRESS(*InBlockY* As String, *InBlockX* As String, *YIntZero_* As _REGRESS_YIntZero__enum, *Labels_* As _REGRESS_Labels__enum, *Confidence* As Double, *SumOutBlock* As String, *Residuals_* As _REGRESS_Residuals__enum, *StdResiduals_* As _REGRESS_StdResiduals__enum, [*ResidualOutBlock* As String], [*ProbOutBlock* As String])

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

InBlockY	Input cells containing a single column of y values (the dependent variables)
InBlockX	Input cells containing one or more columns of x values (the independent variables)
YIntZero	1 if the y-intercept is 0 (the line of regression passes through the origin); 0 if the y-intercept is not 0
Labels	1 if labels are located in the first column or row of the *InBlockY* and *InBlockX*; 0 if the input selections do not contain labels
Confidence	A value indicating the confidence level to apply to the regression
SumOutBlock	Upper-left cell of the output cells for the summary table (allow at least seven columns)
Residuals	1 or 0; if 1, includes residuals in the output table
StdResiduals	1 or 0; if 1, includes standardized residuals in the output table
ResidualOutBlock	Upper-left cell of the output cells for the residuals table (allow at least four columns)
ProbOutBlock	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 *Block*}	Specifies the cells (partial column) containing independent variable (y-axis) data
{Regression.Go}	Performs the regression analysis
{Regression.Independent *Block*}	Specifies cells containing up to 150 columns of independent variable (x-axis) data
{Regression.Output *Block*}	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.

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 *DataToReceive*. You can use this command first to view other choices offered by *AppName*.
"Topics"	A listing of all topics open. For example, a list of open documents under Word for Windows.
"Status"	The current status of the application. For example, READY in Excel or EDIT in Quattro Pro when a cell is being edited.
"Formats"	A list of all Clipboard formats supported by the application or DDE link.
"Selection"	A list of all items currently selected in the application. For example, in Excel cells A3..A47 could be selected.

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

DDEChannel	DDE channel number of the application to receive data from
DataToReceive	Information to receive from the application
DestBlock	Cells to store the data received into

Related topics

{RESIZE}

Syntax

Resize(x As Double, y As Double, *NewWidth* As Double, *NewHeight* As Double, [*VertFlip_* As _Resize_VertFlip__enum], [*HorizFlip_* As _Resize_HorizFlip__enum])

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

x and *y*	XY coordinates of the new upper-left corner, in pixels
NewWidth	The new width, in pixels, of the object or group
NewHeight	The new height, in pixels, of the object or group
VertFlip?	1 if the object or group is flipped vertically from its previous position
HorizFlip?	1 if the object or group is flipped horizontally from its previous position

 **Related topics**

{ResizeToSame}

Syntax

ResizeToSame()

Description

{ResizeToSame} lets you resize selected objects in the dialog window to the same size as the first object selected.

{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 *Block*}	Enters INPUT mode and stays under macro control
{RestrictInput.Exit}	Any operation that ends INPUT mode

{ReturnErrorValue}

Syntax

ReturnErrorValue()}

PerfectScript Syntax

ReturnErrorValue ()

Description

Reinstates the ability for Quattro Pro to return a specific error value, if one is warranted.

{ROWCOLSHOW}

Syntax

RowColShow(*Block* As String, *Show_* As _RowColShow_Show__enum, *Row_* As _RowColShow_Row__enum, *FirstPane_* As _RowColShow_FirstPane__enum)

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

Block	Cells containing rows or columns to hide or show
Show?	1 to reveal rows or columns; 0 to hide rows or columns
Row or *Col*	1 to reveal or hide a row; 0 to reveal or hide a column
FirstPane?	1 to affect rows or columns in left or top window pane; 0 to affect them in the right or bottom window pane

 **Related topics**

{ROWHEIGHT}

Syntax

RowHeight(*Block* As String, *FirstPane_* As _RowHeight_FirstPane__enum, *Reset_* As _RowHeight_Reset__enum, *Size* As Double)

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

Block	Cells containing rows to resize
FirstPane?	1 to resize rows in left or top window pane; 0 to resize rows in right or bottom window pane
Set/Reset	0 to set the row height; 1 to reset the row height
Size	New height (in twips) if setting size; not needed if resetting size

Related topics

{SAMPLE}

Syntax

SAMPLE(*InBlock* As String, *OutBlock* As String, *type* As String, *Rate* As Double)

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

InBlock	One or more numeric or cell values representing the input cells
OutBlock	Upper-left cell of the output cells
Type	"P" to specify periodic sample; "R" to specify random sampling
Rate	A value indicating a sampling rate; if *Type* = "P", Rate indicates the periodic interval used for sampling; if *Type* = "R", Rate indicates the number of samples

Related topics

{SaveHtml_BackgroundColor}

Syntax

SaveHtml_BackgroundColor(*BkColor_* As String)

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_* As String)

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

FileData [semicolon delimited]	Initial name 0 The initial .HTML file name 1 Extension 2 The default extension of .HTML files

{SaveHtml.GraphicType}

Syntax

SaveHtml_GraphicType(*Value_* As Integer)

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

Value	0 The .GIF file format 0 1 The .JPG file format 1 2 The .PNG file format

{SaveHtml_Header}

Syntax

SaveHtml_Header(*Header_* As String)

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_* As String)

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_* As Integer)

PerfectScript Syntax

SaveHtml_Layout (Value?: Numeric)

Description

Lets you specify the layout to be used.

Example

```
SaveHtml.Layout 2
```

Parameter

Value 0 Single page
0 1 Frame enhanced pages
1 2 Multiple pages

{SaveHtml_LineBeforeFooter}

Syntax

SaveHtml_LineBeforeFooter(*Enable_* As _SaveHtml_LineBeforeFooter_Enable__enum)

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
0 1 Footer line

Related topics

{SaveHtml.LineBeforeHeader}

Syntax

SaveHtml_LineBeforeHeader(*Enable*_ As _SaveHtml_LineBeforeHeader_Enable__enum)

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

Enable	0 No header line 0 1 Header line

Related topics

{SaveHtml.LinkColor}

Syntax

SaveHtml_LinkColor(*LinkColor_* As String)

PerfectScript Syntax

SaveHtml_LinkColor (LinkColor?: String)

Description

Lets you specify the default color of the links

Example

```
SaveHtml.LinkColor Black
0 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_* As String)

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_* As Integer)

PerfectScript Syntax

SaveHtml_OutputType (Value?: Numeric)

Description

Lets you specify the type of the output file.

Example

```
SaveHtml.OutputType 1
```

Parameter

Value	0 Output as .HTML 0 1 Output as .XML 1 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_* As String)

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

SourceData [semicolon delimited]	Range 0 The range specified as "Range:" followed by page name, followed by actual range. 1 Boolean 2 0 Output as table 3 1 Output as text

{SaveHtml.TextColor}

Syntax

SaveHtml_TextColor(*TextColor_* As String)

PerfectScript Syntax

SaveHtml_TextColor (TextColor?: String)

Description

Lets you specify the default color of the text.

Example

```
SaveHtml.TextColor Black
0 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(T*i*tle_ As String)

PerfectScript Syntax

SaveHtml_Title (Title?: String)

Description

Lets you specify the title for the file.

Example

```
SaveHtml.Title "Title info of this file"
```

Parameter

Title	The title for the file

{SaveHtml.UseBrowserColor}

Syntax

SaveHtml_UseBrowserColor(*Enable*_ As _SaveHtml_UseBrowserColor_Enable__enum)

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

Enable	0 Lets you specify the colors. 0 1 Use the browser's colors.

Related topics

{SaveHtml.UserDetails}

Syntax

SaveHtml_UserDetails(*UserData_* As String)

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

UserData [semicolon delimited]	Last updated 0 Lets you specify the date of the last update. 1 Updated by 2 Lets you specify who last updated the .HTML file. 3 Email 4 Lets you specify an email address.

{SaveHtml.UseRGBValues}

Syntax

SaveHtml_UseRGBValues(*Enable_* As _SaveHtml_UseRGBValues_Enable__enum)

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

Enable	0 Use the color name strings. 0 1 Use RGB values.

{SaveHtml.Wallpaper}

Syntax

SaveHtml_WallPaper(*Filename_* As String)

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])

0 Scenario_Capture (ScenarioName:String)

1 Scenario_CaptureArea (Area:Enumeration {Notebook!; Page!; Block!; UserDefined!}; [Block:String])

2 Scenario_Close ()

3 Scenario_DeleteGroup (GroupName:String)

4 Scenario_Find ()

5 Scenario_Highlight (Highlight?:Enumeration {Yes!; No!}; ChangeCellColor:Numeric; ResultCellColor:Numeric)

6 Scenario_NewGroup (GroupName:String)

7 Scenario_Open ()

8 Scenario_Remove (ScenarioName:String)

9 Scenario_RemoveCells ([Block:String])

10 Scenario_RenameGroup (OldGroupName:String; NewGroupName:String)

11 Scenario_Report (AllGroups?:Enumeration {Yes!; No!}; LeftLabels?:Enumeration {Yes!; No!}; TopLabels?:Enumeration {Yes!; No!}; [Block:String])

12 Scenario_Show (ScenarioName:String)

13 Scenario_Update_On_Block (Update?:Enumeration {Yes!; No!})

14 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.

0 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}
0 {Scenario.Capture Base Scenario}
1 {Scenario.Update}
2 {SelectBlock A:F4}
3 {PutCell ".096"}
4 {SelectBlock A:C5}
5 {PutCell "60"}
6 {Scenario.Find}
7 {Scenario.Highlight 1,7,9}
8 {Scenario.Capture APR96-60}
9 {Scenario.Update}
10   {SelectBlock A:F4}
11   {PutCell ".085"}
```

```
12   {SelectBlock A:C5}
13   {PutCell "48"}
14   {Scenario.Find}
15   {Scenario.Highlight 1,7,9}
16   {Scenario.Capture APR85-48}
17   {Scenario.Update}
18   {Scenario.Report 0,1,0}
19   {FileSaveAs "C:\COREL\SUITE8\DATA\CARS.WB3"}
20   {Scenario.Close}
```

Options

{Scenario.AddCells <*Block*>}	Defines the selected cells as change-and-result cells.
{Scenario.Capture *ScenarioName*}	Takes a baseline snapshot of data
{Scenario.CaptureArea *Area*,*Block*}	Specifies the area where the Scenario Manager tracks data and format changes.
{Scenario.Close}	Closes a Scenario Manager session.
{Scenario.DeleteGroup *GroupName*}	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 *Highlight?*(0|1), *ChangeCellColor*(0-15), *ResultCellColor*(0-15)}	Turns on and off coloring of change-and-result cells.
{Scenario.NewGroup *GroupName*}	Creates and names a new Scenario Manager group.
{Scenario.Open}	Initializes a Scenario Manager session.
{Scenario.Remove *ScenarioName*}	Deletes the selected scenario.
{Scenario.RemoveCells <*Block*>}	Excludes the selected change-and-result cells from the scenario.
{Scenario.RenameGroup *OldGroupName*,*NewGroupName*}	Applies another name to the active group.
{Scenario.Report *AllGroups*(0|1), *LeftLabels*(0|1), *TopLabels*(0|1), <*Block*>}	Creates a summary report of the change-and-result cells in each scenario.
{Scenario.Show *ScenarioName*}	Lists scenarios you have captured in the active group of scenarios
{Scenario.Update_On_Block *Update?*(0|1)}	Offers options for using the Scenario Manager to track versions.
{Scenario.UseGroup *GroupName*}	Lists the scenario groups included in the active notebook.

{ScenarioExpert}

Syntax

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)

0 Search_Case (Case:Enumeration {Any!; Exact!})

1 Search_Direction (Direction:Enumeration {Row!; Column!})

2 Search_Find (String:String)

3 Search_Look_In (LookIn:Enumeration {Formula!; Value!; Condition!})

4 Search_Match (Match:Enumeration {Part!; Whole!})

5 Search_Next ()

6 Search_Previous ()

7 Search_Replace ()

8 Search_ReplaceAll ()

9 Search_ReplaceBy (String:String)

10 Search_Reset ()

Description

{Search} searches for strings in the active sheet. Use {Search.ReplaceBy} to specify the replacement string; {Search.Replace} replaces the string.

0 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}
0 {Search.Block ""}
1 {Search.Look_In Formula}
2 {Search.Match Part}
3 {Search.Find "1993"}
4 {Search.ReplaceBy "1994"}
5 {Search.ReplaceAll}
```

Options

{Search.Block *Block*}	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 *String*}	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 *String*}	Specifies the group of characters to substitute for characters found.
{Search.Reset}	Clears any entries in the dialog box and

reinstates the defaults.

{SelectAll}

Syntax

SelectAll()

Description

{SelectAll} selects every cell in the active sheet.

{SELECTBLOCK}

Syntax

SelectBlock(*Block* As String, [*ActiveCell* As String])

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

Block	Coordinates of the cell(s) to select
ActiveCell	Address of the cell within the cells to make active

Related topics

{SELECTFLOAT}

Syntax

SelectFloat(*ObjectID* As String, [*MoreObjectID*])

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([*ObjectID* As String], [*MoreObjectID*])

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; [CreateIfNotExist?:Enumeration {Yes!; No!}])

0 Series_Delete (SeriesNumber:Numeric; [AndAllSeriesFollowing?:Enumeration {Yes!; No!}])

1 Series_Go ()

2 Series_Insert (SeriesNumber:Numeric; Block:String)

3 Series_Label_Range (SeriesNumber:Numeric; Block:String; [CreateIfNotExist?:Enumeration {Yes!; No!}])

4 Series_Legend (SeriesNumber:Numeric; LegendText:String)

5 Series_Reverse_Series (Yes?:Enumeration {Yes!; No!})

6 Series_Swap_Row_Col (Yes?:Enumeration {Yes!; No!})

Description

{Series} creates or deletes chart series.

0 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).

1 {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 *CreateIfNotExist?* 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").

2 {Series.Delete}removes an existing series. Set *AndAllSeriesFollowing?* to 1 if you also want to remove all series following *SeriesNumber*.

3 {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.

4 {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 *CreateIfNotExist?* to 1. Then the series will be created if it does not already exist.

5 {Series.Legend}sets the legend text for a series (*LegendTex*t is the new text).

6 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.

7 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}
0 {GraphEdit Profit99}
1 {Series.Data_Range "1",A:A1..A27,1}
2 {Series.Data_Range "2",A:C1..C27,1}
3 {Series.Label_Range "1",A:B1..B27}
4 {Series.Label_Range "2",A:D1..D27}
5 {Series.Data_Range "XAxisLabelSeries",A:E1..E27}
6 {Series.Go}
```

The following macro inserts a new series between the two series in the last example.

```
{GraphEdit Profit99}
0 {Series.Insert 2,A:G1..G27}
1 {Series.Go}
```

Options

{Series.Data_Range *SeriesNumber* | "XaxisLabelSeries" | "LegendSeries", *Block* <,*CreateIfNotExist?* (0|1)>}	Specifies the cell coordinates of the chart data, legend, or label. You must place this value within quotations.
{Series.Delete *SeriesNumber* <,*AndAllSeriesFollowing?*>}	Deletes the selected series.
{Series.Go}	Changes the series according to your selections.
{Series.Insert *SeriesNumber, Block*}	Adds a new series after the selected series.
{Series.Label_Range *SeriesNumber*, *Block* <,*CreateIfNotExist?* (0|1)>}	Specifies the series used for labels.
{Series.Legend *SeriesNumber*, *LegendText*}	Specifies the series used for the legend. You must place this value within quotations.
{Series.Reverse_Series 1|0}	Plots the last series first, then moves backwards through the series order.
{Series.Swap_Row_Col 1|0}	Plots columns as series when Quattro Pro plots series by rows, and plots rows as series when Quattro Pro would plot columns.

{SeriesManager}

Syntax

{SeriesManager.*Option*}

PerfectScript Syntax

SeriesManager_Define (Name:String; FormulaOrList:String; FormulaTextOrRepeat:Any; SeedTextOrValue:String; {[Value:String]})

0 SeriesManager_Go (Name:String; Orientation:Enumeration {Rows!; Columns!; Tabs!}; [Block:String])

1 SeriesManager_Remove (Name:String)

2 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.

0 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"}

0 {SpeedFill}

1 {SeriesManager.Go "First of Month",Columns, A:A2}
```

Options

{SeriesManager.Define *Name*, Formula, *FormulaText*, *SeedText*}	Lets you define a new series.
{SeriesManager.Define *Name*, List, *Repeating?*(0|1), *Value1 <,Value2*, *Value3*,...>}	Lets you define and name a new series.
{SeriesManager.Go *Name*, Rows | Columns | Tabs, *Block*}	Quickly fill cells with a sequence of entries.
{SeriesManager.Remove *Name*}	Deletes the selected series.
{SeriesManager.Rename *OldName*, *NewName*}	Changes the name for the selected series.

Related topics

{SetCellString}

Syntax

SetCellString(*Cell* As String, *String* As String)

PerfectScript Syntax

SetCellString (Cell: String; String: String)

Description

Lets you specify the string insert into the cell.

Example

```
{SetCellString A1; "The string"}
```

Parameters

Cell	The cell into which you want to insert the string
String	The string you want to insert into the cell

{SETGRAPHATTR}

Syntax

SetGraphAttr(*FillColor* As String, *BkgColor* As String, *FillStyle* As String, *BorderColor* As String, *BoxType* As String)

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.

0 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.

1 BoxType specifies the new border style for the object; use any *Border Style* property string included in a chart Object Inspector.

Parameters

FillColor	New fill color of the selected object(s)
BkgColor	New background color of the selected object(s)
FillStyle	New fill style of the selected object(s)
BorderColor	New border color of the selected object(s)
BoxType	New border style of the selected object(s)

Related topics

{SETLCID}

Syntax

SetLCID([*LocalID* As Integer])

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

LocalID	The value of the local ID

{SETMENUBAR}

Syntax

SetMenuBar([*SystemDefinition* As String])

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.

0 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

SystemDefinition	Cells containing a menu system definition

Note

- This command is obsolete.

Related topics

{SETOBJECTPROPERTY}

Syntax

SetObjectProperty(*ObjectProperty* As String, *Value* As String)

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"}

See Property Reference for a list of properties you can use.

Parameters

Object	Object to alter property of
Property	Property to alter
Value	New property setting (or another instance of Object.Property to copy the new setting from)

Related topics

{SETPOS}

Syntax

{SETPOS *FilePosition*}

Description

{SETPOS} moves the file pointer of a file previously opened using OPEN to the value *FilePosition*. *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.

0 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.

1 For an example using {SETPOS}.

Parameters

FilePosition the number of bytes into a file to set the file pointer to

Related topics

{SETPROPERTY}

Syntax

SetProperty(*Property* As String, *Value* As String)

PerfectScript Syntax

SetProperty (Property:String; Value:String)

Description

{SETPROPERTY} alters the properties of the active object (use {SELECTBLOCK}, {SELECTFLOAT}, or {SELECTOBJECT} to select objects).

0 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 (_). See Property Reference for a list of properties you can use.

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

Property	String representing the property to change
Value	String representing the setting to apply to the property

Related topics

{ShowErrorMessage}

Syntax

ShowErrorMessage()

PerfectScript Syntax

ShowErrorMessage ()

Description

Reinstates the ability for Quattro Pro to show an error message, if one is warranted.

Note

- This command is obsolete.

Related topics

{Slide}

Syntax

{Slide.*Option*}

PerfectScript Syntax

Slide_Effect (Effect:String)

0 Slide_Goto (SlideName:String)

1 Slide_Next ()

2 Slide_Previous ()

3 Slide_Run (SlideShowName:String)

4 Slide_Speed (Speed:Numeric)

5 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.Eff ect *Effect*}	Specifies the transition effect to use when displaying the next slide in a slide show.
{Slide.Goto *SlideName}*	Takes the active slide show directly to the slide SlideName.
{Slide.Next}	Advances the active slide show to the next slide.
{Slide.Previous}	Returns the active slide show to the previous slide.
{Slide.Run *SlideShowName}*	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 *Time*}	Specifies the time in seconds to display the next slide in a slide show.

{SlideShowExpert}

Syntax

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)

0 SolveFor_Formula_Cell (Cell:String)

1 SolveFor_Go ()

2 SolveFor_Max_Iters (Iters:Numeric)

3 SolveFor_Reset ()

4 SolveFor_Target_Value (Value:Numeric)

5 SolveFor_Variable_Cell (Cell:String)

Description

{SolveFor} solves goal-seeking problems with one variable.

0 {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.

1 {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.

2 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!})

0 Sort_Block (Block:String)

1 Sort_Data (Order:String)

2 Sort_Go ()

3 Sort_Heading (Heading?:Enumeration {Yes!; No!})

4 Sort_Key_1 (Cell:String)

5 Sort_Key_2 (Cell:String)

6 Sort_Key_3 (Cell:String)

7 Sort_Key_4 (Cell:String)

8 Sort_Key_5 (Cell:String)

9 Sort_Labels (Use:String)

10 Sort_Order_1 (Order:String)

11 Sort_Order_2 (Order:String)

12 Sort_Order_3 (Order:String)

13 Sort_Order_4 (Order:String)

14 Sort_Order_5 (Order:String)

15 Sort_PreviousSorts (PreviousSorts?:Numeric)

16 Sort_Reset ()

17 Sort_Type (Type?:String)

Description

{Sort} sorts the entries in cells. To perform the sort, use {Sort.Go} after the other sort command equivalents.

0 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}
0 {Sort.Block "A:A3..C40"}
1 {Sort.Type Top to bottom}
2 {Sort.Heading 0}
3 {Sort.Key_1 a25}
4 {Sort.Key_2 c23}
5 {Sort.Order_1 Ascending}
6 {Sort.Order_2 Ascending}
7 {Sort.BlankCellsFirst No}
8 {Sort.Data Numbers First}
9 {Sort.PreviousSorts -1}
10   {Sort.Labels Dictionary}
11   {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*}	Clears all entries and restores defaults.
{Sort.*Type* "Left to Right" | "Top to Bottom"	Determines to sort by rows or columns.

{SPEEDFILL}

Syntax

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.

0 To create or modify a series used with QuickFill, use {SeriesManager.*Option*}.

Related topics

{SpeedFormat}

Syntax

SpeedFormat(*FmtName* As String, *NumFmt_* As _SpeedFormat_NumFmt__enum, *Font_* As _SpeedFormat_Font__enum, *Shading_* As _SpeedFormat_Shading__enum, *TextColor_* As _SpeedFormat_TextColor__enum, *Align_* As _SpeedFormat_Align__enum, *LineDraw_* As _SpeedFormat_LineDraw__enum, *AutoWidth_* As _SpeedFormat_AutoWidth__enum, *ColHead_* As _SpeedFormat_ColHead__enum, *ColTotal_* As _SpeedFormat_ColTotal__enum, *RowHead_* As _SpeedFormat_RowHead__enum, *RowTotal_* As _SpeedFormat_RowTotal__enum, [*SubTotals_* As _SpeedFormat_SubTotals__enum])

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.

0 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.

1 To add or remove formats, use {SpeedFormat.*Option*}.

Parameters

FmtName	Name of the format to apply
NumFmt?	1 to apply the numeric format; 0 otherwise
Font?	1 to apply the font; 0 otherwise
Shading?	1 to apply the shading; 0 otherwise
TextColor?	1 to apply the text color; 0 otherwise
Align?	1 to apply the alignment; 0 otherwise
LineDraw?	1 to apply the line drawing; 0 otherwise
AutoWidth?	1 to automatically size the columns; 0 otherwise
ColHead	1 to apply the column heading format; 0 otherwise
ColTotal?	1 to apply the column total format; 0 otherwise
RowHead?	1 to apply the row heading format; 0 otherwise
RowTotal?	1 to apply the row total format; 0 otherwise
SubTotals?	1 to apply the subtotal format; 0 otherwise

Related topics

{SpeedFormat}

Syntax

{SpeedFormat.*Option*}

PerfectScript Syntax

SpeedFormat_Add (Name:String; ExampleBlock:String)

0 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

{SpeedFormat.Add *Name*, *ExampleBlock*}	Creates a new custom format.
{SpeedFormat.Remove *Name*}	Deletes the active SpeedFormat.

Related topics

{SPEEDSUM}

Syntax

SpeedSum([*Block* As String]

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}

Syntax

STEP()

Description

{STEP} is equivalent to the Debug key, Shift+F2.

Note

- This command is obsolete

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.

{TABLE}

Syntax

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 *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.

Examples

The following macro searches the external table TASKLIST.DB using the query file TASKLIST.QBE. The results of the search are stored in A:A2.

```
{TableQuery.FileQuery Yes}
{TableQuery.QueryFile TASKLIST.QBE}
{TableQuery.Destination A:A2}
{TableQuery.Go}
```

The next macro searches the same database, but uses the query defined in the named cell task_query.

```
{TableQuery.FileQuery No}
{TableQuery.QueryBlock task_query}
{TableQuery.Destination A:A2}
{TableQuery.Go}
```

Options

{TableQuery.Destination *Block*}	Specifies the cells for the query's Answer Table (its results).
{TableQuery.FileQuery Yes|No}	Specifies an external query file as the source of the query text.
{TableQuery.Go}	Performs the table query.
{TableQuery.QueryBlock *Block*}	Specifies cells in the active notebook as the source of the query text.
{TableQuery.QueryFile *Filename*}	Specifies the filename or cell coordinates of the query text.

Related topics

{TableView}

Syntax

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.

{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.
{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.

{UNDO}

Syntax

UNDO

Description

{UNDO} "takes back" the last command given and restores the previous state for most commands.

Related topics

{UngroupObjects}

Syntax

UngroupObjects()

Description

{UngroupObjects} separates the selected group of chart annotation objects so each can be moved or modified without affecting the others.

{VLINE}

Syntax

VLine(*Distance* As Integer)

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.

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* As Integer)

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 method PGUP to move the selector vertically.

Parameters

Distance	Number of screens to scroll the active notebook vertically

Related topics

{WebQuery_Create}

Syntax

WebQuery_Create(*FileData_* As String)

PerfectScript Syntax

WebQuery_Create (Filedata?: String)

Description

Lets you create a new Web query file.

Parameter

Filedata	The name of the new query file

{WebQuery.Destination}

Syntax

WebQuery_Destination(*DestRange_* As String)

PerfectScript Syntax

WebQuery_Destination (DestRange?: String)

Description

Lets you specify the output location. If empty, a new sheet will be used.

Parameter

DestRange	The range of cells

{WebQuery_LinkRange}

Syntax

WebQuery_LinkRange(*LinkRange_* As String)

PerfectScript Syntax

WebQuery_LinkRange (LinkRange?: String)

Description

Lets you specify the range of cells to be associated with Web link.

Parameter

LinkRange	The range of cells

{WebQuery_LinkRefreshDuration}

Syntax

WebQuery_LinkRefreshDuration(*Value_* As String)

PerfectScript Syntax

WebQuery_LinkRefreshDuration (Value?: String)

Description

Lets you specify the refresh duration in seconds

Parameter

Value	Must be in the format "hh:mm:ss"

{WebQuery.LinkRefreshTime}

Syntax

WebQuery_LinkRefreshTime(*Time_* As String)

PerfectScript Syntax

WebQuery_LinkRefreshTime (Time?: String)

Description

Lets you specify the value of start time, end time, start day, and end day.

Parameter

Time	Must be in the format "hh:mm:ss"

{WebQuery.LinkRefreshType}

Syntax

WebQuery_LinkRefreshType(*LinkRefreshOptions_* As String)

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

LinkRefreshOptions [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_ As String)

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

LinkWrapOptions [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

FileName	The name of the query file.

{WebQuery.Run}

Syntax

WebQuery_Run()

PerfectScript Syntax

WebQuery_Run ()

Description

Lets you run the current query.

{WebQuery.SetQueryOptions}

Syntax

WebQuery_SetQueryOptions(*QueryOpts_* As String)

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

QueryOpts [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_* As String)

PerfectScript Syntax

WebQuery_SetQueryParameters (QueryParams?: String)

Description

Lets you specify the parameter value options.

Parameters

QueryParams [semicolon delimited]	Parameter Parameter type Parameter value

{WebQuery.Source}

Syntax

WebQuery_Source(*SourceRange_* As String)

PerfectScript Syntax

WebQuery_Source (SourceRange?: String)

Description

Lets you specify the range of cells to be updated from the query output.

Parameter

Range	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 *Block*}	Specifies the cells where you want to write the data table.
{WhatIf.Input_Cell_1 *Cell*}	Specifies the first (or only) cell referenced by the what-if formula.
{WhatIf.Input_Cell_2 *Cell*}	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

{WindowArrIcon}

Syntax

WindowArrIcon()

Description

{WindowArrIcon} lines up minimized windows on the Quattro Pro desktop or icons on the Objects sheet.

{WindowCascade}

Syntax

WindowCascade()

Description

{WindowCascade} rearranges all open windows on the Quattro Pro desktop.

{WindowClose}

Syntax

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}

Syntax

WindowHide()

Description

{WindowHide} conceals the active notebook window.

{WindowMaximize}

Syntax

WindowMaximize

Description

{WindowMaximize} is equivalent to Maximize in a Control menu, which enlarges the active window so it fills the screen.

{WindowMinimize}

Syntax

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()

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

UpperLeftX	Distance between the left side of the Quattro Pro window and the left side of the active window, in pixels
UpperLeftY	Distance between the bottom of the input line and the top of the active window, in pixels

{WindowNewView}

Syntax

WindowNewView()

Description

{WindowNewView} displays a duplicate copy of the active notebook in a new window.

{WindowNext}

Syntax

WindowNext()

Description

{WindowNext} is equivalent to choosing Next in a Control menu. It makes the next window active.

{WindowPanes}

Syntax

WindowPanes(*Mode* As _WindowPanes_Mode_enum, Synch_ As _WindowPanes_Synch__enum, [Width As Double], [*Height* As Double])

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

Synch?	Whether the panes are synchronized: yes (1) or no (0)
Width	Width of the left pane or height of the upper pane (optional)
Height	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}

Syntax

WIndowRestore()

Description

{WindowRestore} is equivalent to Restore on the Control menu. It restores minimized windows to their original size.

{WindowShow}

Syntax

WindowShow(*Name* As String)

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

Name	Name of the hidden window to show

{WindowSize}

Syntax

WindowSize(*x* As Double, *y* As Double)

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

X	New window width, in pixels
Y	New window height, in pixels

{WINDOWSOFF}

Syntax

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_name         General Cement Co.
                     Alveoli Mfg., Inc.
                     Sandab Development
                     Consolidated Dust
```

Related topics

{WINDOWSON}

Syntax

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}

Syntax

WindowsTile()

Description

{WindowTile} displays all open windows without overlapping them.

{WindowTile.TileTopToBottom}

Syntax

WindowTile_TopToBottom()

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* As String])

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 ()

Description

{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 *Filename}*	Overlays any existing windows with the windows stored in the workspace file, then retrieves the appropriate file for each.
{Workspace.Save *Filename}*	Saves the position and size of all notebook windows and the names of the files contained in each window.

{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

Block	The Block
LabelsTop	0 1
LabelsLeft	0 1
LabelsBottom	0 1
LabelsRight	0 1
Intersection	0 1

{XMLTag.Create}

Syntax

{XMLTag.Create *TagName; Block*}

PerfectScript Syntax

XMLTag_Create (TagName: String; Block: String)

Description

Equivalent to Insert  XML Tag...
 Generate...

Parameters

TagName
Block

{XMLTag.Delete}

Syntax

{XMLTag.Delete *TagName*}

PerfectScript Syntax

XMLTag_Delete (TagName: String)

Description

Equivalent to Insert  XML Tag...
 Delete...

Parameter

TagName

{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

InBlock1	One or more numeric cell values representing the first input cells
InBlock2	One or more numeric cell values representing the second input cells
OutBlock	Upper-left cell of the output cells
Labels	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
Alpha	Significance level of the test; the default is 0.05
Difference	A value indicating the hypothetical difference in the means between InBlock1 and InBlock2; the default is 0
Variance1	A value indicating the variance of data set one; the default is 0
Variance2	A value indicating the variance of data set two; the default is 0

Note

- This command is obsolete

Related topics

Numeric Format Codes

Code	Description
0-15	Fixed (0-15 decimals)
16-31	Scientific (0-15 decimals)
32-47	Currency (0-15 decimals)
48-63	% (percent; 0-15 decimals)
64-79	, (comma; 0-15 decimals)
112	+/- (bar 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)

Quattro Pro VBA Events

Visual Basic for Applications (VBA) is an event-driven programming language. Most of the code you create is written to respond to an event. An event is an action that is recognized by VBA; for example, clicking a button or choosing an option from a list box. Unlike traditional procedural programming, in which the program starts at line 1 and executes line by line, event-driven programming executes code in response to events.

All events in Quattro Pro 10 are code placeholders. It is up to you to code the response. All events are called in response to a specific action. When an action occurs, the appropriate event will be called and the code located within the event is executed. You can create simple or complex events. You can code a single line that displays a Message Box or write an entire procedure that interacts with a database.

All events are members of the **Document** class. The name of the object is the same as the class. There are four events in Quattro Pro.

Quattro Pro Events

- AfterOpen()
- BeforeSave()
- AfterSave()
- BeforeClose()

Document.BeforeSave()

Syntax

```
Private Sub Document_BeforeSave()
```

Description

This event is called just before the Quattro Pro notebook is saved. This gives you a chance to customize your Quattro Pro notebook before you save it.

Example

In the following example, the numeric values are added and the result is written to the appropriate cell.

```
Private Sub Document_BeforeSave()

'*** Calculate the January totals
PerfectScript.SelectBlock "B2 B4"
PerfectScript.QuickFunction "SUM", "B5"

'*** Calculate the February totals
PerfectScript.SelectBlock "C2 C4"
PerfectScript.QuickFunction "SUM", "C5"

'*** Calculate the March totals
PerfectScript.SelectBlock "D2 D4"
PerfectScript.QuickFunction "SUM", "D5"
PerfectScript.SetCellString "A5", "Tot

End Sub
```

Note

- The code which created the inventory table is entered in the Document.AfterOpen().

Document.AfterSave()

Syntax

```
Private Sub Document_AfterSave()
```

Description

This event is called after you have saved your Quattro Pro notebook.

Example

In the following code fragment, a Message Box appears with the time and date. This data can be stored to a database which keeps track of file activities.

```
Private Sub Document_AfterSave()
'*** Declare all variables
Dim myTime
Dim myDate As Date
Dim myStrTime, myStrDate, Msg As String

'**** Populate the variables
myTime = Time
myDate = Date

myStrDate = Str(myDate)
myStrTime = Str(myTime)

'*** Display the Message Box
Msg = "The date is " & myStrDate & " and the time is " & myStrTime
MsgBox Msg
End Sub
```

Document. BeforeClose()

Syntax

```
Private Sub Document_BeforeClose()
```

Description

This event is called when the Quattro Pro notebook is closed; however, this code is executed before the notebook is actually closed.

Example

In the following code example, a Message Box will inform the user that the Quattro Pro notebook will close. This Message Box will appear before the notebook is closed.

```
Private Sub Document_BeforeClose()
MsgBox "You are about to close this document"
End Sub
```

Document.AfterOpen()

Syntax

```
Private Sub Document_AfterOpen()
```

Description

This event is called when the Quattro Pro document opens.

Example

You can customize your Quattro Pro notebook by adding a table. When the notebook opens, the following code will produce an inventory table:

```
Private Sub Document_AfterOpen()
'******* Create a Table
PerfectScript.SetCellString "B1", "Jan"
PerfectScript.SetCellString "C1", "Feb"
PerfectScript.SetCellString "D1", "Mar"
PerfectScript.SetCellString "A2", "TVs"
PerfectScript.SetCellString "A3", "VCRs"
PerfectScript.SetCellString "A4", "Radios"

'****** Populate the January Column
PerfectScript.SelectBlock "B2"
PerfectScript.PutCell2 "200"
PerfectScript.SelectBlock "B3"
PerfectScript.PutCell2 "250"
PerfectScript.SelectBlock "B4"
PerfectScript.PutCell2 "350"

'****** Populate the February Column
PerfectScript.SelectBlock "C2"
PerfectScript.PutCell2 "100"
PerfectScript.SelectBlock "C3"
PerfectScript.PutCell2 "280"
PerfectScript.SelectBlock "C4"
PerfectScript.PutCell2 "340"

'*** Populate the March Column
PerfectScript.SelectBlock "D2"
PerfectScript.PutCell2 "150"
PerfectScript.SelectBlock "D3"
PerfectScript.PutCell2 "230"
PerfectScript.SelectBlock "D4"
PerfectScript.PutCell2 "490"
End Sub
```

VBA Programming Issues Relating to Macro Commands

There are several issues that must be discussed with respect to programming with product commands in the VBA environment. You can click on any of the following gray boxes for a detailed explanation:

- Product commands with **Repeating Parameters**
- Product commands that require a **VARIABLE**

Product commands with repeating parameters

To use product commands in VBA with repeating parameters, you must declare an array. Values for each repetitive parameter must be loaded into the array. After the array is populated, you have to pass the array The following list is all the macro commands in Quattro Pro with repeating parameters:

Product commands with repeating parameters:

- ExecAuto
- DLL
- Delvar
- SelectFloat
- SelectObject
- CrossTab
- GraphView
- CreateObject
- Code Example

Working with repeating parameters

You must create and pass an array to each product command that has repeating parameters. Refer to the following code example, which illustrates how to use **PreTaskBar**:

Example 1

```
'***** Create the variables
Dim boxes As Variant
Dim widths As Variant
Dim textIcon As Variant

'****** Populate each array
boxes = Array(7, 9, 2, 4, 10, 5)
textIcon = Array(0, 0, 0, 0, 0, 0)
widths = Array(50, 75, 50, 20, 100, 50)

'****** Pass each array
PerfectScript.PrefTaskBar boxes, textIcon, widths
```

Example 2

```
****** Populate each array
PerfectScript.PrefTaskBar Array(7, 9, 2, 4, 10, 5),Array(0, 0, 0, 0, 0, 0),Array(50, 75, 50, 20, 100, 50)
```

Code Explanation

You must define the box style, the icon style, and the width for every item that you want to appear on the application bar. In the example above, there are six elements in each array, meaning that six items will appear on the application bar. Each element corresponds to the item. The **boxes** array defines the box style for each item. All values in the **textIcon** array are 0, meaning there will be no icons in any of the items. The values in the **widths** array specifies the width for each item Notice that **PreTaskBar** has **boxes**, **textIcon**, and **widths** as arguments.

In the second example, all the Arrays were populated during the product command call. The benefit of this method is it decreases the lines of code in your macro.

Note

- You must use the integer values when populating an array used for repeating parameters.

Presentations product commands that use a VARIABLE as a parameter

You must use a Variant for any product command that requires a variable as a parameter. The Variant data type is the data type for all variables that are not declared as another specific type. If you do not declare the variable as a Variant, then your VBA macro will not function properly . The following list is all the product commands which use a Variable as a parameter:

Product commands that use a VARIABLE as a parameter:

- PutCell2
- Let
- Put
- FloatCreate
- PutCell
- RecalcCol
- Random
- PutBlock2
- ReCalc
- PutBlock
- Code Example

Working with product commands that use a Variable

You must declare a variable that you pass to a product command as a Variant.
Refer to the following code fragment:

```
'**** Declare the variable
Dim myAnswer As Variant

'*** Pass the variable to DirectoryExists()
PerfectScript.DirectoryExists myAnswer, "D:\Client"
MsgBox myAnswer
```

Code Explanation

A Boolean value is returned to **myAnswer**. If the directory exists, then **myAnswer** will be assigned the value *True*. If the directory does not exist, then **myAnswer** will be assigned *False*.

Event

Each object within an object model is defined by a property, method, event, or a combination of each. An event is a noun, and acts as something that takes place in an object. You write code for an object to respond to the act. Events are triggered by an action, such as a click, key press, or system timer.

Event-driven programming

Visual Basic for Applications is an event-driven programming language. Most of the code you create is written to respond to an event. Each object within an object model is defined by a property, method, event, or a combination of each. An event is a noun, and acts as something that takes place in an object. You write code for an object to respond to the act. Events are triggered by an action, such as a click, key press, or system timer. Unlike traditional procedural programming, in which the program starts at line 1 and executes line by line, event-driven programming executes code in response to events.

Variant

The Variant data type is the data type for all variables that are not declared as another type such as Dim, Private, Public, or Static. The Variant data type has no type-declaration character.

Object-oriented programming

A style of programming that places emphasis on creating and using objects.

Object model

An object model represents the hierarchy of objects within an application and their relationship to each other within the paradigm.

For example, the **Document** object represents the beginning of the object hierarchy in WordPerfect. Starting with the Document object, you drill down and navigate through the object model until you find the desired object. To reference an object with Visual Basic code, you separate each level of the object hierarchy with the dot operator (.).

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 runing 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 FieldCmpItemPreset 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 FieldCmpItemPreset 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 FieldSumary 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 agaist 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	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 MoveCell 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 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	**$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 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.

Visual Basic for Applications and WordPerfect Office

Visual Basic for Applications (VBA) is an object-oriented programming language that lets you create VBA macros to automate tasks. You can, for example, create a macro in WordPerfect that changes the color of the headings. WordPerfect Office includes version six of the Microsoft Visual Basic for Applications (VBA) programming language.

VBA is an event-driven programming language. Most of the code you create is written to respond to an event. An event is an action that is recognized by VBA; for example, clicking a button or choosing an option from a list box. Unlike traditional procedural programming, in which the program starts at line 1 and executes line by line, event-driven programming executes code in response to events.

All events in the application are code placeholders. It is up to you to code the response. All events are called in response to a specific action. When an action occurs, the appropriate event will be called and the code located within the event is executed. You can create simple or complex events. You can code a single line that displays a message box or write an entire procedure that interacts with a database.

Getting Started with VBA

What is Visual Basic for Applications?

Visual Basic for Applications (VBA) is a subset of the Microsoft Visual Basic (VB) object-oriented programming environment. VBA uses the Visual Basic Editor interactive development environment and the VB programming language to enhance applications by manipulating the application's objects, exposed by its object model. VBA can access other applications by referencing that application's object model components.

WordPerfect Office includes version six of the Microsoft Visual Basic for Applications (VBA) programming language. VBA is a subset of the Microsoft Visual Basic (VB) object-oriented programming environment. VBA uses the Visual Basic Editor interactive development environment and the VB programming language to enhance applications by manipulating the application's objects, exposed by its object model. VBA is a standard programming language that allows you to customize the application for your needs and integrate Corel products with other VBA-enabled applications by referencing that application's object model components.

VBA provides you with a set of tools that you can use to customize the graphical user interface of Corel applications. These tools allow you to process information and present data in an efficient and effective forum. Developers using VBA to extend Corel applications will benefit from the familiar Visual Basic language, Rapid Application Development (RAD) integrated development environment, and fast runtime performance in the resulting integrated solutions. Developers will also benefit from an extensible forms package that supports ActiveX controls for creating user interfaces, access to the full Windows API and the underlying file system, connectivity to corporate data, and integration with other COM-based software.

Even though VBA uses the Visual Basic programming language, it is considered "for applications" because it is most often integrated into another application in order to customize the functionality of that application.

 **Related topics**

What is Event driven programming?

Visual Basic for Applications is an event-driven programming language. Most of the code you create is written to respond to an event. Each object within an object model is defined by a property, method, event, or a combination of each. An event is a noun, and acts as something that takes place in an object. You write code for an object to respond to the act. Events are triggered by an action, such as a click, key press, or system timer. Unlike traditional procedural programming, in which the program starts at line 1 and executes line by line, event-driven programming executes code in response to events.

Related topics

What is the difference between Visual Basic, Visual Basic for Applications and VBScript?

The Microsoft Visual Basic programming system is an advanced set of programming tools that provides advanced functionality and components for the Microsoft Windows operating system and other windows-based programs. For example, with Visual Basic you can create application extensions (dll's) and stand-alone executable programs (exe's). You cannot create either of these components with VBA or VBScript.

VBA is also referred to as Visual Basic, Applications Edition. VBA is a subset of the Visual Basic programming language. It uses the programming structure of Visual Basic to manipulate objects of an object model, left exposed by an application. The manipulation of these objects results in small packets of code procedures within the application. These code procedures and resulting projects are called add ins.

VBScript is also referred to as Microsoft Visual Basic, Scripting Edition. VBScript is also a subset of the Visual Basic programming language. It is a web-based HTML document scripting language.

Related topics

What is the difference between VBA and PerfectScript?

Previously, you could only use the PerfectScript language to automate specific tasks. Both product commands and programming commands are used in conjunction with the PerfectScript language. The PerfectScript language is useful for developing simple macros. VBA offers more flexibility and power. When you use VBA to create macros, you are assisted by the Visual Basic compiler. The compiler helps you by providing context-sensitive help when you are coding a VBA macro. You can combine the power of VBA with the PerfectScript product commands to create powerful macros. You have to use the Visual Basic Editor to create VBA macros; however, PerfectScript macros are developed from the WordPerfect Editor. You can access the Visual Basic Editor only when you are working in an active document.

Related topics

Working in the VBA Editor

When you work in the VBA Editor, you can create a new object, such as a dialog box, which is known as a form. You can add controls, such as a check box or a text box. You can set the object's properties in the Property dialog box. You can also set the object's properties at run time by programming a method.

Each document that you create with VBA has a corresponding Visual Basic for Applications project. In order to customize your document with VBA coding procedures, you must open the project file in the Visual Basic Editor. To display the Editor, go to **Tools|Visual Basic|Visual Basic Editor** on the main menu in the application.

For more detailed information on constructing code procedures and setting properties, see the Microsoft Visual Basic Help in the Visual Basic Editor.

Related topics

Using VBA macros

VBA allows you to edit and play macros that automate a series of tasks within an application.

You can store a VBA macro in the document by saving the document. Once you have saved the document, you can close and reopen the document and access the macro. After you have developed the macro, you should debug it. You can step through each macro line by line. This is a useful exercise to ensure that the macro will have the desired outcome. A project macro is not available if the document is closed. After you have debugged the macro, you can play the macro.

For more detailed instruction relating to VBA and its programming environment, please consult the "Microsoft Visual Basic for Applications Help" from the Help menu in the Visual Basic Editor.

Related topics

Accessing an Application from another Application's macro

You can access and change an application from another application's macro. VBA uses the Visual Basic Editor interactive development environment and the VB programming language to enhance applications by manipulating the application's objects, exposed by its object model. VBA allows you to customize your needs and integrate Corel products with other VBA-enabled applications by referencing that application's object model components.

For example, you could create and use a Quattro Pro object from a WordPerfect VBA macro. This allows you to change and save a Quattro Pro document from a WordPerfect VBA macro.

 **Related topics**

Quattro Pro VBA Macros Help

Click the Help Topics button to return to the list of topics.

Using ActiveX Components

An ActiveX component (*.OCX) enables you to add a great amount of power and flexibility to your VBA macro. An ActiveX component is basically a special type of DLL (dynamic link library). Originally, ActiveX components were created to replace Visual Basic controls, however they have exceeded this purpose. Visual Basic for Applications (VBA) is an ActiveX container, meaning that you can include ActiveX components in your VBA macro.

The components which are located on the toolbar are part of the Microsoft 2.0 Object library. These components are meant for VBA programming. You can add additional components to your VBA project. However, some components may work and others may not. Not all ActiveX components are meant for the VBA environment. It is recommended that you became familiar with a individual component before you add it to your VBA macro.

If you are trying to add a new Active X control to your VBA Toolbox and are receiving errors stating that the control is not licensed or that the control just does not work properly, this is not a bug - here may be many reasons for these error messages:

- Many Windows applications write Active X controls for their own use and therefore are not supported or even expected to be used by other applications. Many of the controls that are included with Corel WordPerfect Office are of this nature and cannot be used with VBA.
- Some Active X controls installed to your system may have been included with other development applications such as Visual C++, Visual Basic, Delphi etc, and they may have license requirements that only allow them to run in their own development environment. Therefore, they will not work with VBA.

As a result, only those Active X controls available with Microsoft Forms 2.0 that are shipped as part of Microsoft Visual Basic for Applications 6.0 are supported in WordPerfect Office. Any others you have on your system may be used with VBA, but may not be actually intended for use in this manner, and therefore will not work. Even if they do work, you may not have rights to distribute them to your VBA Macro users.

if you are using custom controls, be very careful that the control you are using is meant to be used in VBA and that you have the proper licensing rights required for its use in your application.

To add an ActiveX component to your VBA Form

1. From the VBA Editor, select Insert, User Form.
2. Select Tools, Additional Controls
3. Select the desired component.

Quattro Pro PerfectScript Class Members

A
B
C
D
E
F
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

- AbsoluteReference
- ACTIVATE
- ADDMENU
- ADDMENUITEM
- ADDSERIES
- AddSubMenuItem
- Alert
- AnalysisExpert
- ANOVA1
- ANOVA2
- ANOVA3
- Application.Property
- Audit.Remove_All_Arrows
- Audit.Trace_Dependents
- Audit.Trace_Precedents

B

- BEEP
- BLANK

C

D

E

F

G

H

I

J

No macros.

K

No macros.

L

M

N

O

P

- Page.Property
- PageViewGoto
- PANELOFF
- PANELON
- ParseExpert_ApplyFormatting
- ParseExpert_CellDelimiterString
- ParseExpert_CellDelimiterTypeComma
- ParseExpert_CellDelimiterTypeMultiSpace
- ParseExpert_CellDelimiterTypeOther
- ParseExpert_CellDelimiterTypeReturn
- ParseExpert_CellDelimiterTypeSemiColon
- ParseExpert_CellDelimiterTypeSpace
- ParseExpert_CellDelimiterTypeTab
- ParseExpert_ColumnWidths
- ParseExpert_ConsecutiveAsOne
- ParseExpert_DataType
- ParseExpert_DelimiterType
- ParseExpert_Go
- ParseExpert_IgnoreNonConformingRows
- ParseExpert_InputBlock
- ParseExpert_InputFile
- ParseExpert_InputType
- ParseExpert_JoinBrokenLines
- ParseExpert_LineLength
- ParseExpert_LoadSettings
- 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_SaveSettings
- ParseExpert_SettingsFile
- ParseExpert_SheetDelimiterString
- ParseExpert_SheetDelimiterTypeComma
- ParseExpert_SheetDelimiterTypeMultiSpace
- ParseExpert_SheetDelimiterTypeOther
- ParseExpert_SheetDelimiterTypeReturn
- ParseExpert_SheetDelimiterTypeSemiColon
- ParseExpert_SheetDelimiterTypeSpace
- ParseExpert_SheetDelimiterTypeTab
- ParseExpert_Skip1stChar
- ParseExpert_TextQualifier
- ParseExpert_ValueQualifier
- PasteSpecial
- POKE

Q

R

S

X

Y

No macros.

Z

No macros.