






About Aligning Controls

Use the Location and Size options on the Properties dialog box to position and size controls precisely using dialog units. You can also click and drag to size and position controls manually.

Use the options on the Align menu to position and space controls inside the dialog and in relation to each other. The last control you click is the "anchor control," which appears with black squares around it.

-  [To align and space controls](#)
-  [To edit controls](#)
-  [To add controls](#)
-  [To delete controls](#)
-  [**Related topics**](#)

To align and space controls,

- 1 Press **Shift** while clicking the control(s) you want to align with each other, then click **Align ▶ Left, Right, Top,** or **Bottom** to move the control to that side of the anchor control.
- 2 Press **Shift** while clicking the control(s) you want to align with each other, then click **Align ▶ Center** ▶ **Vertical** or **Horizontal** to center a control in relation to the anchor control.
- 3 Press **Shift** while clicking the control(s) you want to make the same size as the anchor control, then click **Align ▶ Make Same Size** ▶ **Vertical, Horizontal,** or **Both**.
- 4 Press **Shift** while clicking the controls you want to space in relation to each other, then click **Align ▶ Space Evenly** ▶ **Vertical** or **Horizontal**.

Tips

- ◆ The anchor control is the last control you select.

 [Related topics](#)


About Bitmap

Display a bitmap as a control. The bitmap appears on the background of the dialog without a border. You can use horizontal and vertical lines to outline it for clarity.



A bitmap control looks like this:

Use the Properties dialog to set the location, size, named regions, variables, and other properties for the control.


For information about controls on the dialog box, click , then click a control.

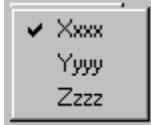
- [To add controls](#)
- [To edit controls](#)
- [To delete controls](#)
- [Related topics](#)

About Buttons

There are several different kinds of buttons you can add to a dialog box.

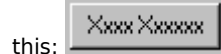
Pop-up buttons display a list of options when clicked. The button itself shows the selected option. A pop-up

button looks like this when it is closed  , and like this



when you click it.

A push button activates a specific action when clicked, such as OK, Cancel, Help, and so on. A push button looks like








this:

Radio buttons represent mutually exclusive options. Selecting one radio button deselects another. A radio button

looks like this: 


Use the Properties dialog to set the location, size, named regions, variables, and other properties for each control.

For information about controls on the dialog box, click , then click a control.

-  [To add controls](#)
-  [To edit controls](#)
-  [To delete controls](#)
-  [Related topics](#)

About Center


Use Center to center a control in relation to the anchor control. Press Shift to select multiple controls. The last control you select is the anchor control, and all the selected controls will be centered in relation to it. You can also press Ctrl while selecting a control to center a copy of it.

 [To align and space controls](#)


 [Related topics](#)





About Check Box

Check boxes represent compatible options. Clicking an empty check box selects the option; clicking a marked check box deselects the option. You can also define check boxes as "Three State" to give a third value, where a box is checked and grayed to be unavailable.

A check box looks like this:  A three-state check box with an empty square on the left, a checked square in the middle, and a grayed-out square on the right.

Use the Properties dialog to set the location, size, named regions, variables, and other properties for the control.

For information about controls on the dialog box, click , then click a control.

-  [To add controls](#)
-  [To edit controls](#)
-  [To delete controls](#)
-  [**Related topics**](#)


About Color Wheel

Color wheels allow users to select colors based on values of hue, lightness, and saturation.



A color wheel looks like this:

Use the Properties dialog to set the location, size, named regions, variables, and other properties for the control. A keyboard interface has been added to the date controls. The arrow keys may be used to move the color selection. Hold down the CTRL key while using the arrows to change the value of the color saturation bar.

For information about controls on the dialog box, click , then click a control.



To add controls



To edit controls



To delete controls



Related topics

About Combo Box

Combo boxes display an edit box and a list box. Enter text in the edit box, or double-click a list item to insert it.


A combo box looks like this  when it is closed and like this



when you click it.

Use the Properties dialog to set the location, size, named regions, variables, and other properties for the control.



For information about controls on the dialog box, click , then click a control.



To add controls



To edit controls



To delete controls



Related topics

About Control Groups

Use Control Groups to group controls together. Defining a group box around a set of controls does not group them, and you must set the control order before grouping controls.



To set up dialogs



Related topics

About Control Order

Use Control Order to specify the sequence in which you want to tab through the controls. You must set the control order before grouping the controls.



To set up dialogs



Related topics

About the Control Toolbar

Use the Control toolbar to choose controls and actions in the PerfectScript Dialog Editor.



To add controls



To copy controls



To delete controls



To edit controls



To position multiple controls



To position single controls



Related topics

To add controls,



- 1 Click **Tools** **Dialog Editor**.
- 2 Select the macro you want, then click **OK**.



- 3 Double-click the name of an existing dialog, or click **File** **New** to create a dialog.
- 4 Click **Control**, then click the control you want to add.
- 5 Position the "x" above the control symbol where you want the top-left corner of the control to appear, then click.
- 6 Double-click the control, set its properties, then click **OK**.
- 7 Repeat steps 4-6 for the rest of the controls you want to add.

Tips

- ◆ You can also add controls using the icons on the toolbar.



[Related topics](#)

To copy controls,



- 1 Click **Tools** **Dialog Editor**.
- 2 Select the macro you want, then click **OK**.
- 3 Double-click or create a dialog with controls on it.
- 4 Select the control(s) you want to copy.
- 5 Right-click, then click **Copy** from the QuickMenu.
- 6 Drag the copy to position it.

Tips

- ◆ The copy has all the properties of the original control. Double-click the new control to change them.
- ◆ You can also press **Ctrl** while selecting to copy a control.



Related topics

To delete controls,



- 1 Click **Tools** **Dialog Editor**.
- 2 Select the macro you want, then click **OK**.
- 3 Double-click or create a dialog with controls on it.
- 4 Select the control you want to delete, then press **Del**.



[Related topics](#)

To edit controls,



- 1 Click **Tools** **Dialog Editor**.
- 2 Select the macro you want, then click **OK**.



- 3 Double-click the name of an existing dialog, or click **File** **New** to create a dialog.
- 4 Select the control you want to edit.
- 5 Double-click the control, make the changes you want to its properties, then click **OK**.
- 6 Click and drag each control to position and size it.



[Related topics](#)

To position multiple controls,



- 1 Click **Tools** **Dialog Editor**.
- 2 Select the macro you want, then click **OK**.
- 3 Double-click or create a dialog with controls in it.
- 4 Click the top-left corner of the top control in the group you want to select.
- 5 Drag the selection area down and right to select all the controls you want in the group.
- 6 Drag the grouped controls to position them.

Tips

- ◆ You can also hold down **Shift** while clicking controls to select multiple controls.



Related topics

To position single controls,



- 1 Click **Tools** **Dialog Editor**.
- 2 Select the macro you want, then click **OK**.
- 3 Double-click or create a dialog with controls in it.
- 4 Double-click a control, specify a left and top value to position it on the dialog, specify a width and height value to change its size, then click **OK**.

Tips

- ◆ You can also click and drag each control to position and size it.



[Related topics](#)



About PerfectScript Dialog Editor






















Use the PerfectScript Dialog Editor to quickly and easily create, design, set properties for, and edit the dialogs you use in your macros. Using the Dialog Editor takes the place of the DialogDefine and DialogAdd sections in your macros.

The Dialog Editor also lets you add, edit, position, size, move, and assign values to controls, lines, and other elements of a dialog. You can use it to give each control a control name, variable name, values, and other properties, as well as define the control order, set the initial focus, set tab stops, and group controls.





Most dialogs that are part of the macro system (not user-defined macro dialogs) now allow the static text message part of the dialog to be copied to the clipboard.



For information about controls on any dialog box, click  , then click a control.

 	<u>To copy dialogs</u>
 	<u>To create new dialogs</u>
 	<u>To delete dialogs</u>
 	<u>To dismiss a dialog if you are using a callback</u>
 	<u>To display a dialog within your macro</u>
 	<u>To edit dialogs</u>
 	<u>To open the PerfectScript dialog editor from WordPerfect</u>
 	<u>To rename dialogs</u>
 	<u>To save new and edited dialogs in the current macro</u>
 	<u>To select macros and dialogs for editing</u>
 	<u>Related topics</u>

To copy dialogs,

- 1 Click **Tools**  **Dialog Editor**.
- 2 Select the macro you want, then click **OK**.
- 3 Select the dialog you want to copy.
- 4 Click **Edit**  **Copy**.
- 5 Open the macro file you want to copy the dialog to, then click **Edit**  **Paste**.
- 6 Select the copy, click **File**  **Rename**, then type a new name for the dialog.



Tips

- ◆ Copy places an exact copy of a dialog in another macro file or in the current macro file.



[Related topics](#)

To create new dialogs,

- 1 Click **Tools**  **Dialog Editor**.
- 2 Select the macro you want to add a dialog to, then click **OK**.
- 3 Click **File**  **New**.
- 4 Type a name for the new dialog, then click **File**  **Open** to edit the dialog.
- 5 Add controls, choose a font, and set the properties for the dialog.
- 6 Click **File**  **Save** to save the new dialog in the current macro file, then click **File**  **Close** to exit the dialog.

Tips

- ◆ Be sure to use the same names for the dialogs that you used in your macros, with the same case, since dialog and control names are case sensitive.



[Related topics](#)

To delete dialogs,



- 1 Click **Tools** **Dialog Editor**.
- 2 Select the macro you want, then click **OK**.
- 3 Select the dialog you want to delete.



- 4 Click **File** **Delete**.
- 5 Click **Yes**.

Tips

- ◆ Delete removes a dialog from a macro file, but you must also delete all references to a deleted dialog from the macro itself using your selected macro editor.



[Related topics](#)

To dismiss a dialog if you are using a callback,

- 1 In your macro editor, type or insert the DialogDismiss command after a DialogShow command that uses a callback.
- 2 Specify the name of the dialog and the name of the control used to dismiss the dialog.

Tips

- ◆ If you use a cancel button, a control other than a push-button, or a non-existent control to dismiss the dialog, your changes will not take effect. If you use a push-button other than a cancel button, the variable values are set and your changes take effect when you dismiss the dialog.



Related topics

To display a dialog within your macro,

- 1 In your macro editor, type or insert the DialogShow command where you want the dialog to appear in your macro.
- 2 Specify the name of the dialog and the named region for its parent window.
- 3 Specify a callback parameter if you want the macro to execute while the dialog displays.
- 4 Type or insert the DialogDismiss command after a DialogShow command that uses callback.
- 5 Specify the name of the dialog and the name of the control used to dismiss the dialog.



[Related topics](#)

Example of DialogShow and DialogDismiss

The following is an example of the DialogShow command:

```
DialogShow ("DialogName"; "WordPerfect"; Callback@)
```

The first parameter is the name you gave to the dialog when you created it in the PerfectFit Dialog Editor (in this case, DialogName).

The second parameter is a named region specifying the parent window for the macro dialog (in this case, the WordPerfect window will be the parent window for the dialog). Named regions are defined by the application. The region consists of the application name, followed by a period (.), followed by additional words that narrow the named region to the appropriate window. For example, the named region of the document window in WordPerfect is **WordPerfect.Document**.

The third parameter is a label that identifies a callback function. If you do not specify a callback parameter in the DialogShow command, the macro does not execute until you dismiss the dialog. If you use a callback, the macro executes while the dialog is up. It is up to the callback to prevent the macro from terminating prematurely and to shut down the macro dialog using the DialogDismiss command.

The following is an example of the DialogDismiss command:

```
DialogDismiss ("DialogName"; "OKBttn")
```

The first parameter is the dialog name. The second parameter is the named region of the control used to dismiss the dialog. For more information on the DialogShow and DialogDismiss commands, see Macros Online Help.



Related topics

To edit dialogs,



- 1 Click **Tools** **Dialog Editor**.
- 2 Select the macro file containing the dialog you want to edit, then click **OK**.
- 3 Double-click the dialog you want to edit.
- 4 Add, edit, or delete controls, change the font, and alter the properties for the dialog.



- 5 Click **File** **Save** to save the edited dialog in the current macro file.



[Related topics](#)

To open the PerfectScript dialog editor from WordPerfect,

  **1** Click **Tools** **Macro**

  **Edit.**

2 Select a macro filename, then click **Edit** to display the Macro toolbar.

3 Click **Dialog Editor** on the Macro toolbar.

4 Select the dialog you want, then click **Edit**.

  **Related topics**

To rename dialogs,



- 1 Click **Tools** **Dialog Editor**.
- 2 Select the macro you want, then click **OK**.
- 3 Select the dialog you want to rename.



- 4 Click **File** **Rename**.
- 5 Type a new name for the dialog.

Tips

- ◆ Renaming gives a dialog a new name while leaving it in the current macro file, but it does not change the name displayed on the caption bar.
- ◆ When you rename a dialog, you will need to change all references to it in the macro itself using your selected macro editor.



Related topics

To save new and edited dialogs in the current macro,

- 1 Click **Tools**  **Dialog Editor**.
 - 2 Select the macro you want, then click **OK**.
 - 3 Double-click the name of an existing dialog, or click **File**  **New** to create a dialog.
 - 4 Add, edit, or delete controls, choose the font, and set the properties for the dialog.
 - 5 Click **File**  **Save**, then click **File**  **Close** when you are finished with the dialog.
-  [Related topics](#)

To select macros and dialogs for editing,



- 1 Click **Tools** **Dialog Editor**.
- 2 Select the macro file(*.wcm) you want to create or edit dialogs for, then click **OK**.



- 3 Select the dialog you want to edit, or click **File** **New** to add a new dialog to the macro.

Tips

- ◆ The PerfectScript Dialog Editor works only with macros in WordPerfect format. The Dialog Editor does not allow you to edit the macros themselves, only to define dialogs for them.



[Related topics](#)

About Counter


Counters allow users to enter numeric data in an edit box by typing or by clicking an incrementor/decrementor. Clicking inserts a number in the edit box that is within a specified range.

A counter looks like this:



Use the Properties dialog to set the location, size, named regions, variables, and other properties for the control.



For information about controls on the dialog box, click , then click a control.



To add controls



To edit controls



To delete controls




Related topics

About Create/Edit List

Use Create/Edit List to enter and change options for controls that use lists, such as pop-up lists, drop-down lists, list boxes, and so on.



For information about controls on the dialog box, click  , then click a control.



To add and edit list items



To create new dialogs



Related topics

To add and edit list items,

- 1 Double-click a control that uses a list of options.
- 2 Click **Create/Edit List**.
- 3 Type a name, then choose **Add** for each option in the list.
- 4 Select an item in the list, type a new name, then click **Replace** to edit an item.
- 5 Select an item in the list, then click **Move Up** or **Move Down** to reposition an item.
- 6 Select an item on the list, then click **Delete** to remove list items.
- 7 Select an item in the list, then click **Set Initial** to have that item selected when the dialog opens.

Tips

- ◆ You can also select the **Sort list** check box to sort the list items alphabetically. When Sort list is selected, you can not rearrange items in the list.



Related topics

About Custom


Custom controls let you specify your own options for text, class, and attributes.

A custom control looks like this:



Use the Properties dialog to set the location, size, named regions, variables, and other properties for the control.



For information about controls on the dialog box, click , then click a control.



To add controls



To choose control properties



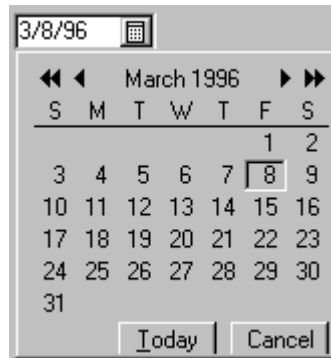
To edit controls



Related topics

About Date

Date controls display an edit box and a calendar icon. Users can enter dates by typing or by clicking the calendar icon to open a calendar and selecting a date to display in the edit box.



A date control looks like this when you click the calendar icon:


Use the Properties dialog to set the location, size, named regions, variables, and other properties for the control. You can use the following key combinations to change the date quicker:

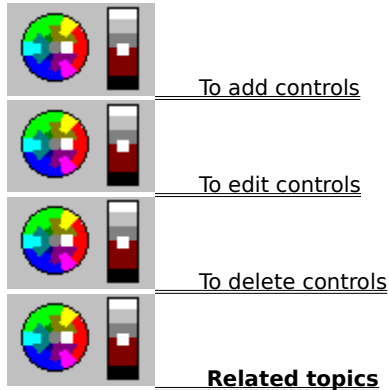
arrow + ctrl - Increases the tens column.

arrow + alt - Increases the hundreds column.

A keyboard interface has been added to the date controls. Pressing the ALT key and the arrow keys will change the month by 1 month. The PageUp/PageDown keys are used to change the years. Using the ALT key will change the year by 1 year. With ALT and CTRL, change the year by 10 years, and with ALT and SHFT, change the year by 100 years.



For information about controls on the dialog box, click , then click a control.



About Default Button

Use Default Button to specify the button you want to be activated when the user presses Enter on the control.



To set up dialogs



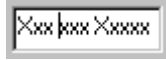
To add controls



Related topics

About Edit Box


Edit boxes allow users to type text or allow the macro to type text for them. They can have one line or multiple lines.



An edit box looks like this:

Use the Properties dialog to set the location, size, named regions, variables, and other properties for the control.



For information about controls on the dialog box, click , then click a control.



To add controls



To edit controls



To delete controls



Related topics

About Filename Box


Filename boxes display an edit box and a folder icon list button. Users can enter filenames or directories by typing them in or by clicking the list button to display a Browse dialog.

A filename box looks like this:



Use the Properties dialog to set the location, size, named regions, variables, and other properties for the control.



For information about controls on the dialog box, click  , then click a control.



To add controls



To edit controls



To delete controls



Related topics

About Font

Use Font to choose a typeface and point size for all text on your dialog. The caption font remains constant for all dialogs. Changes in font size and style do affect the size of the dialog.






To choose fonts for dialogs



Related topics

To choose fonts for dialogs,

- 1 Click **Tools**  **Dialog Editor**.
- 2 Select the macro you want, then click **OK**.
- 3 Double-click the name of an existing dialog, or click **File**  **New** to create a dialog.
- 4 Click **Dialog**  **Font**.
- 5 Select a font and point size.

Tips

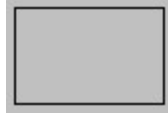
- ♦ The font size you choose will affect the size of your dialog--larger font means larger dialog area.



[Related topics](#)

About Frame


Frames can visually group items in a dialog, or act as design elements.



A frame looks like this:

Use the Properties dialog to set the location, size, named regions, variables, and other properties for the control.



For information about controls on the dialog box, click , then click a control.



To add controls



To edit controls



To delete controls



Related topics



About Group Box









Group boxes visually group controls in a dialog with a titled frame. Group boxes do not automatically group the controls, however. Use Control Groups and Control Order to group controls.



A group box looks like this:
Use the Properties dialog to set the location, size, named regions, variables, and other properties for the control.



For information about controls on the dialog box, click  , then click a control.

-   To add controls
-   To edit controls
-   To delete controls
-   **Related topics**

About Initial Focus

Use Initial Focus to specify the control that has the focus when the dialog opens.




To set up dialogs



Related topics

About Lines

Horizontal and vertical lines visually separate items in a dialog.


A horizontal line looks like this: 



A vertical line looks like this:

Use the Properties dialog to set the location, size, named regions, variables, and other properties for the control.



For information about controls on the dialog box, click  , then click a control.



To add controls



To edit controls



To delete controls



Related topics

About List Box


List boxes display lists of options to choose from.



A list box looks like this:

Use the Properties dialog to set the location, size, named regions, variables, and other properties for the control.



For information about controls on the dialog box, click  , then click a control.



To add controls



To edit controls



To delete controls



Related topics

About Make Same Size

Use Make Same Size to make a selected control the same size as the anchor control. Press Shift to select multiple controls. The last control you select is the anchor control, and all the selected controls will be made the same size as it is.



To add controls



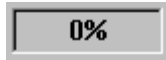
To choose control properties



Related topics

About Progress Indicator


Progress indicators display the progress of a process as it runs.



A progress indicator looks like this:

Use the Properties dialog to set the location, size, named regions, variables, and other properties for the control.



For information about controls on the dialog box, click , then click a control.



To create new dialogs



To edit controls



To delete controls




Related topics

About Properties

Use Properties to specify the location and size of the dialog, its caption, class, dialog type, frame type, and attributes.

Use the Properties dialogs associated with each control to set the location, size, named regions, variables, and other properties for each control.



For information about controls on the dialog box, click  , then click a control.



To add controls



To choose control properties



To choose dialog properties



To create new dialogs



Related topics

To choose control properties,



- 1 Click **Tools** **Dialog Editor**.
- 2 Select the macro you want, then click **OK**.
- 3 Double-click or create a dialog with controls in it.
- 4 Double-click a control, then make the changes you want to its properties.

Tips

- ◆ You can also click and drag controls to position and size them.



Related topics

To choose dialog properties,



- 1 Click **Tools** **Dialog Editor**.
- 2 Select the macro you want, then click **OK**.



- 3 Double-click the name of an existing dialog, or click **File** **New** to create a dialog.
- 4 Right-click the dialog, then click **Properties** from the QuickMenu.
- 5 Specify the location, size, caption, help file and help key, attributes, dialog type, and frame for the dialog.



[Related topics](#)

About Scroll Bars

Scroll bars allow users to scroll from left to right and top to bottom.




A vertical scroll bar looks like this:



A horizontal scroll bar looks like this:

Use the Properties dialog to set the location, size, named regions, variables, and other properties for the control.



For information about controls on the dialog box, click  , then click a control.



To add controls



To edit controls



To delete controls




Related topics

About Setting Up Dialogs

Use the options on the Dialog menu to group dialog controls, to specify the default focus and buttons, and to define tab stops. You must set the control order before grouping the controls; defining a group box around a set of controls does not automatically group them.



For information about controls on any dialog box, click , then click a control.



To add controls



To set up dialogs



Related topics

To set up dialogs,



1 Click **Tools** **Dialog Editor**.

2 Select the macro you want, then click **OK**.

3 Double-click or create a dialog with controls in it.



4 Click **Dialog** **Control Order**, click the controls in the order you want to tab through them, then click **OK**.



5 Click **Dialog** **Control Groups**, click each control to select its group, then click **OK**.



6 Click **Dialog** **Default Button**, click a control, click the button you want to be activated when the user presses Enter on the control, then click **OK**.



7 Click **Dialog** **Initial Focus**, click the control you want to have as the focus when the dialog displays, then click **OK**.



8 Click **Dialog** **Tab Stops**, click to highlight the controls you want to tab to, then click **OK**.



[Related topics](#)

About Space Evenly

Use Space Evenly to space controls horizontally or vertically inside the dialog and in relation to each other.




To align and space controls



Related topics


About Static Text

Static text on a dialog gives instructions or information to users. Static text can consist of one line or many lines, and is read only.

Static text looks like this: 

Use the Properties dialog to set the location, size, named regions, variables, and other properties for the control.



For information about controls on the dialog box, click , then click a control.



To add controls



To edit controls



To delete controls



Related topics

About Tab Stops

Use Tab Stops to select the controls you want to tab to.



To set up dialogs



Related topics

About Testing a Dialog

Use Test to check your dialogs before closing the PerfectScript Dialog Editor. Testing a dialog lets you check its controls to make sure they appear and function correctly before you close the Dialog Editor. When you are finished testing, click an OK or Cancel button, or click the Close button on the dialog's title bar.



To save new and edited dialogs in the current macro







To create new dialogs



Related topics

To test dialogs before closing the Dialog Editor,

- 1 Click **Tools**  **Dialog Editor**.
- 2 Select the macro you want, then click **OK**.
- 3 Double-click the name of an existing dialog, or click **File**  **New** to create a dialog.
- 4 Add, edit, or delete controls, choose the font, and set the properties for the dialog.
- 5 Click **Dialog**  **Test**.
- 6 Use each control on the dialog, and make changes as needed.
- 7 When you are finished testing, click **File**  **Save**, then click **File**



About Using the Grid to Design Dialogs

Use the options on the View menu to align controls with precision. By displaying a grid of dotted lines on your dialog, you can neatly position and line up controls. You can also force controls to stick to grid lines with Snap to Grid and select grid spacing intervals for even greater precision placement.



To align and space controls



To position controls with a grid



Related topics

To position controls with a grid,



1 Click **Tools** **Dialog Editor**.

2 Select the macro you want, then click **OK**.

3 Double-click or create a dialog with controls in it.



4 Click **View** **Show Grid**.



5 Click **View** **Snap to Grid** to force controls to align with grid points.



6 Click **View** **Grid Options**, specify the amount of space you want between grid points on each axis, then click **OK**.

7 Add controls to the dialog using the grid as a guide.



[Related topics](#)

About Viewer


Viewers display read-only, scrollable text files.



A viewer looks like this:

Use the Properties dialog to set the location, size, named regions, variables, and other properties for the control.



For information about controls on the dialog box, click , then click a control.



To add controls



To edit controls



To delete controls



Related topics

To customize macro toolbar buttons,

- 1 Select **Tools, Settings**, then click the **Toolbar** tab.
- 2 Choose the macros that you want associated with each button.

Tips

- The PerfectScript toolbar displays three buttons that can be configured to play any macro that you choose.
- A Windows shell menu item of "Debug" has been added for macros. It is displayed on the context menu for a desktop icon for a macro.
- The toolbar in PerfectScript and the Macro Debugger is shown as a flat toolbar, similar to other applications within the WordPerfect Office suite.



Related topics

To record macros in JavaScript,

- 1 Select **Tools, Settings**, then click the **Record** tab.
- 2 Select "JavaScript" as the script language.



[Related topics](#)

Removing source from macros

When removing source from macros in PerfectScript, the macros are changed so that they cannot be accidentally edited in WordPerfect. In addition, you can now remove source from macros compiled from previous versions of PerfectScript.

When you remove source from a macro, the Options dialog displays the option to remove the source or protect the macro from being accidentally edited in WordPerfect. The Options dialog also stores and displays the current state of the macro (i.e. if source code has been removed from the macro or if the macro is marked as protected).

The protected macro file type is accepted for all commands. Once a macro is marked as protected, it may be compiled and played, but not opened in WordPerfect. Any attempt to open a protected macro in WordPerfect will generate an unknown file type error. The operation of removing source code from a macro cannot be reversed, and this item may appear disabled in the Options dialog. However, marking a macro as protected can be reversed, and this option is always enabled in the Options dialog.



Related topics

Selecting files with the Open/Save dialog

When you use the Open/Save dialog to select a file, the position and size of the dialog are recorded so that when you return to it, it maintains its size and position during the current session of PerfectScript.



Related topics

Shift and help clicking in the Command Inserter

Shift clicking (holding down the shift key and clicking) and help clicking (selecting the "?" button in the caption bar and clicking) are now easier to use in the Commands list box. In previous versions of PerfectScript, the command specific help was invoked for the item already selected and the item clicked on was not selected first. As a result, in order to get the command specific help for a command in the Commands list box, you had to first select the desired command and then shift click or help click.

This pre-selection is now combined into the Shift clicking and help clicking in order to eliminate any extra steps and confusion that accompanied the previous method of invoking the command specific help.

Tips

- Selecting **SHIFT|CTRL|ALT** when clicking a command in command listbox displays the return enumerations (if any) for that command.
- When command names, parameters names, or enumerations are too wide for the listbox, a horizontal scrollbar displays for easier navigation of the list items.
- Click the Return Values button in the Command Inserter to return value enumerations for those commands that have enumerations as return values.
- The name of the application attached to the Command Inserter is displayed in the caption bar of the Command Inserter.
- The command list for the application attached to the Command Inserter is initially displayed when the Command Inserter is opened.



Related topics

Using default values with Parameters of commands

For most optional enumeration parameters of commands, a default value is passed when the parameter is left off and in some cases, leaving off the optional enumeration parameter performs a different function. The default enumeration values, if any, for these parameters is now displayed in bold text in the Enumerations list box.

There are some cases where the default value is a combination of enumerations. In this case, several enumerations may be defined with the same value as synonyms for each other. All enumerations defined with the same value as the default value are highlighted.



Related topics

About Breakpoints

Use Breakpoints to indicate locations in the macro file where execution of the macro should be interrupted and suspended, and where the Debugger should become active to allow you to examine state of the macro. You can specify the conditions under which you want the Debugger to stop the macro as it plays and bring up the Debugger dialog box. You can make the macro stop at any line, .DLL call, or other place in a macro. The Debugger dialog will then display the labels, functions, procedures, variables being passed, and so on, so you can check them. All breakpoints allow a message to be logged to the Debugger's Event Log.

Each individual breakpoint may be disabled. When disabled, a breakpoint is ignored even if its conditions occur. Additionally, all breakpoints may be temporarily disabled. This causes no breakpoints to be recognized. This is temporary, and all breakpoints revert to their previous state when the temporary disable is ended, or when the macro ends and the Debugger terminates.

The Debugger automatically creates three breakpoints: Macro Start, Macro End, and Error. It marks the breakpoints it adds with an !, and you can remove these if you want to.

The Breakpoints dialog has a context menu that allows breakpoints to be sorted by various columns.


These are the types of breakpoints you can use:

- **DLL Call:** Breaks when your macro calls a .DLL file.
- **Error:** Breaks when an error occurs while running the macro.
- **Label/Routine Call:** Breaks when a label or user-defined routine comes up in the macro. Everything that is not specifically contained in a routine or label is in the <main> routine.
- **Label/Routine Return:** Breaks when a label or user-defined routine comes up in the macro, but stops before executing the return from the label call.
- **Line Number:** Breaks when the macro reaches a line number you specify. This is the most common type of breakpoint. It suspends and brings up the Macro Debugger when execution of the macro reaches a specified line number.
- **Product Call:** Breaks when the macro makes a call to an application or product..
- **Variable Access:** Breaks when the macro accesses a variable.
- **Variable Assign:** Breaks when the macro assigns a value to a variable.

If you do not specify a different macro file, the breakpoint will apply to all the macro files used by the macro you are debugging. If you want to limit the breakpoint to a single macro file, specify that file along with the other breakpoint information.

Most breakpoints allow you to specify a passcount as well as other information. A passcount is the number of times that the breakpoint conditions can occur before the breakpoint actually takes effect. Each time the conditions occur, the passcount decrements, and when the count reaches zero, the breakpoint triggers and causes a break into the Debugger.



For information about controls on the dialog box, click  , then click a control.



About the Event Log



To make a debugger log file



To search macro source code



To specify where you want the Debugger to stop while playing a macro



Related topics

To enable and disable breakpoints,

- 1 Right-click a line with a breakpoint, then click **Enable/Disable All Breakpoints** to toggle breakpoints on and off for the entire macro at once.

Tips

- Right-click a line with a breakpoint, then click **Disable Breakpoint** to force the Debugger to ignore an individual Line Number breakpoint.
- Right-click a disabled breakpoint, then click **Enable Breakpoint** to allow the Debugger to see the breakpoint again.
- To remove a breakpoint, right-click the line with the breakpoint you want to delete, then click **Remove Breakpoint**.




[Related topics](#)

To specify where to stop the Debugger while playing a macro,

 **1** Click **File** **Debug**

 **Play.**

2 Specify the macro you want to play with the Debugger.

3 When the Debugger dialog box opens, click **Debug**  **Breakpoints**

 **Edit.**

4 Select the type of breakpoint you want to use, then click **Add**.

5 Specify the options you want to use for the type of breakpoint you added, then click **Update**.

Tips

- ◆ Each type of breakpoint has different options. Press **Shift+F1**, then click a control for specific information.
- ◆ You can add a Line Number breakpoint easily by double-clicking the line in the macro source list where you want the macro to pause.

 **Related topics**

About Compiling Macros

A macro compiler is used to format macros so that applications can play them. This is called compiling. Macros in WordPerfect Office applications are compiled when you play or record them.

If you receive an error message while the macro is compiling, you can choose Cancel Compilation to close the dialog box, or you can choose Continue Compilation to check for other errors that may be in the macro. In either case, the macro will not play. You must correct all errors, then recompile the macro to play it.

The Cancel button on the Compile Progress dialog is the default button on the dialog. This allows the user to cancel the compilation of a macro by hitting the Enter key, without having to use the mouse or tab key to shift focus to the Cancel button.


To set compile settings, click **Tools|Settings** and select the **Compile** tab. The "Warn when using unsupported features" option defaults to True.

Note

The following conditions will return a compile error:

- No "return" statement is found in the body of a user-defined function. A "return (0)" statement is generated by the Compiler.
- A "return" statement with no return value is found in the body of a user-defined function. A value of 0 is returned.
- The macro appears to be empty when compiled. This occurs because an empty macro could be a previously compiled macro which has had its source removed. Compiling such a macro will destroy the existing compiled macro object.
- An obsolete or unsupported feature is found in a macro during compilation. These warnings can be safely ignored and produce a successful macro - they serve as reminders only. Warnings are displayed when an old EN English synonym is used in the Application statement (US, UK, CE, OZ), or when an obsolete/unsupported command, enumeration, or parameter is used. There is a setting on the Compiler page that allows the user to enable/disable these warnings. This setting is initially enabled.



For information about controls on the dialog box, click , then click a control.



To convert a macro to WordPerfect format



To compile macros



Related topics

To compile macros,



- 1 Click **File** **Compile**.
- 2 Select the macro you want to compile.

Tips



- ◆ While compiling a macro, you can click **File** **Stop Compile** if you need to halt the compile process.
- ◆ If you are compiling an older macro or one in a different format, you will need to convert it into a more current format.
- ◆ If your macro references labels and routines not defined inside it, you will get a warning when the compiler reaches that point. As long as the labels and routines are defined in another macro that the current macro calls, there will not be a problem running the macro later.
- ◆ You can also compile macros by selecting a macro file name in the Explorer, then clicking **Compile** from the QuickMenu.




[Related topics](#)

About Convert Macro to WordPerfect Format

Use Convert Macro to WordPerfect Format to convert a macro to a format your Corel suite applications can use. You can replace the macro or save it under a different filename or in a different directory.



For information about controls on the dialog box, click , then click a control.



To convert a macro to WordPerfect format



Related topics

To convert a macro to WordPerfect format,



- 1 Click **File** **Play**.
- 2 Select the macro you want to convert.
- 3 Specify a path and filename for the converted macro.



Related topics

About PerfectScript



PerfectScript is an application that you can use to record, play, compile, convert, and edit macros and to build or edit dialog boxes for macros. A macro is a series of commands and menu selections in a file that can be replayed using a few keystrokes or a mouse click.

Use macros to record commands that perform tasks automatically. For example, you can create a macro that retrieves a file, gives it a new name, and saves it in another format.

Use About PerfectScript for information about your version number, the release date, program paths, and memory usage.

PerfectScript checks system registry settings as part of its startup. If incorrect entries are found, a warning is displayed and, if the user has write access to the system registry, PerfectScript notifies the user before correcting the system registry entries.



For information about controls on any dialog box, click  , then click a control.



Related topics

To exit PerfectScript,

1 Click **File**   **Exit.**



Related topics

About PerfectScript Commands

Use PerfectScript Commands to select, edit, and insert macro commands into a macro. PerfectScript Commands gives you access to application-specific commands (such as commands that function only in Corel Presentations) and commands that are common to all WordPerfect suite applications. The PerfectScript Command Inserter allows you to easily insert product commands into a macro you are building.

Many macro commands use parameters, which are variables used with a command to indicate a specific value or option.

Product commands perform functions that let you use suite applications features in your macros. Many product commands require you to specify parameters and value set members; these give applications information about options to choose in dialog boxes and specify whether certain features (such as the Ruler) should be displayed or hidden. You can use the Command Inserter to specify the commands quickly.

Programming commands allow you to control how application features act and interact. For example, you can use programming commands to request information from the user or specify that part of a macro run several times. The Command Inserter displays the proper syntax of the programming command you select.


When you use the Command Inserter, you can spend less time typing and worrying about spelling errors. You simply choose the commands and parameters you want from the list boxes and insert them into your macro.

For more information on individual macros, select the macro you want from the Commands list box, then right-click the command.



To insert macro commands,



- 1 Click **File**  **Edit**.
- 2 Select the macro you want to work on, then click **Edit**.
- 3 Move the insertion point to the place in the macro where you want to insert the commands.
- 4 Click **Commands** on the Macro Property Bar to open the Command Inserter.
- 5 Select the type of command you want to insert, then select and double-click a command.
- 6 Select a parameter, type in or select the value for the parameter, then click **Update**.
- 7 Double-click enumerations to add both the parameter and enumerations.
- 8 Click **Insert** to place the new command in the macro.

Tips

- ◆ You can edit a parameter by selecting the parameter and changing its value in the Command Edit text box.
- ◆ Before you can use the Command Inserter, use the Edit tab on the Settings dialog box to specify WordPerfect as the default macro editor.
- ◆ If you are using the Command Inserter that has a Parameter Value text box, you can select the command to insert, then type in a value for the parameter. Clicking **Value** lets the application automatically insert a value for the parameter, and clicking Update changes the display to reflect the new value.



Related topics

About Corel Web Site

If you have installed a World Wide Web browser, and have a modem or network connection to an Internet service



provider, you can click **Help** **Corel Web Site** (www.corel.com) and go to the Corel home page for WordPerfect Office.

The home page contains links to:

- Technical Information Documents from Customer Services
- Hints and tips, demos, and other task-oriented resources



Related topics

About Create New Variable

Use Create New Variable to add a new local, global, or persistent variable to the macro you are debugging.



To create new variables



To delete a variable while debugging a macro



Related topics

To create new variables,

1 Click **File**  **Debug** 

 **Play.**

2 Select a macro you have compiled for debugging, then click **Play**.

3 When the Debugger stops at the place you want to add a variable, click **Variables**

4 Type a name for the variable, click a variable type, then click **Create**.



[Related topics](#)

About Debugging Macros

Use the PerfectScript Debugger to play and compile macros to find errors. The main benefit of the macro is that it helps you eliminate errors in your macros by providing access to information as a macro executes. The Debugger provides information both about the contents of variables and the flow of execution in a macro.

You can go straight through macros, step by step, or set specific breakpoints for checking variables. By setting the appropriate types of breakpoints in the appropriate places, and by examining of the information provided by the Debugger, you can usually narrow down and determine the cause of macro problems. In addition, since the Debugger lets you modify the normal flow of execution when needed, the Debugger can also allow you to make some temporary corrections to the current state of a macro to bring the actual state back in alignment with the expected state.

To begin debugging a macro, first use the Debugger to compile the macro, and any other macro file that it Uses or Runs or Chains to, to include debugging information, and to generate a compiler listing file that the Debugger can use. As the Debugger compiles a macro, it numbers each line of the macro. These numbers appear beside each line of the source code of the macro, which displays as the macro plays. The added lines of source code make the macro a bit larger, but make it possible for you to debug it more easily. In order to see the macro source code in the Debugger, you need to select the "Include debug information" check box on the Compile tab in the Settings dialog box.

Because the compiler does not format the code in your macro the same way a WordPerfect suite application does, you may want to generate a list file when you compile your macro. The list file makes a copy of each procedure and numbers each line, so you can tell which part of the macro corresponds to each line number. It also lists error messages and warnings that came up during the compile at the bottom of the file. Using the list file makes it easy to find and correct errors that the Debugger catches.

You can debug a macro directly with PerfectScript. You can also select options in the Settings dialog box that allow you to automatically compile macros for debugging from any application in the Suite.

Restarting Macros

The Macro Debugger has the ability to restart a macro while it is running. This capability is assigned to the SHIFT-F5 accelerator key. When a macro is restarted, all variables created by the macro, except Persistent variables, are deleted and the Debugger will begin at the top of the macro. All other state information is reset to the initial conditions so that when the macro is restarted, it is as if the macro has not yet been run. A macro can be restarted once it has finished running.

Displaying Variables

At each breakpoint, the Macro Debugger displays the contents of the current variables in the variable list, including the values of array variables, where the contents of all the elements are displays in an abbreviated list. But when an array has a large number of elements, the display of all elements is time consuming and of little practical value, so only the value of the first 25 elements are displayed. However, the values of all array elements are still visible by expanding the array.

Expanding an Array in the Variable List

Expanding the contents of an array in the Macro Debugger can consume a great deal of time and memory. To speed up the process, only the first 100 elements of an array are initially displayed. When the array is expanded, a warning prompt is displayed, asking if all elements should be expanded, or just the first 100 elements. An additional 101st element is also displayed, representing the remaining elements in the array, without their values. The remaining elements can be displayed by clicking the [...] icon next to the 101st element, or by collapsing the array and expanding it again, which prompts the same warning, allowing the user to select the option that expands all elements.

Debugger Config File

When the debugger config file is loaded, the version number check is for \leq the current version (the version number of the macro system differs slightly from the version number of the PerfectScript shared code). When a macro is debugged, a common config file, STARTUP.DBG, and its commands, are loaded before the config file for the macro.

The following items are recorded and restored in the Debugger config file when a macro is debugged:


- Macro listing filename is saved with macro filenames in the MRU list
- Which info windows are open
- Which variable pools are displayed











Refreshing the Variable List

Whenever the Macro Debugger stops at a breakpoint, the contents of all variables are refreshed and displayed. When a macro contains a large number of variables, this process can be quite lengthy, so the Debugger can be configured to display on the first few variables, which speeds up the time it takes to refresh the list. The Debugger will initially display the first 100 variables, but at any time, the contents of all variables can be

displayed by selecting the menu item "Show All Variables".



For more information about controls on the dialog box, click , then click a control.

-  [About the Debugger Interface](#)
-  [About Moving Through Macros While Debugging](#)
-  [To choose debugging options for macros](#)
-  [To debug macros](#)
-  [To edit macros](#)
-  [To generate a listing file for debugging macros](#)
-  [To search macro source code](#)
-  [To select debugging options](#)
-  [Tips for Debugging Macros](#)
-  [**Related Topics**](#)


To choose debugging options for macros,


1 Click **File**  **Debug**

 **Play.**

2 Select the macro you want to debug.


3 Click **Debug**  **Breakpoints**

 **Edit**, set the breakpoints you want in the macro, then click **Close**.

4 Click **View**  **Variables**, then select the types of variables you want to display.

5 Click **Variables**  **New** to create new variables.

6 Select a variable, then click **Variables**  **Delete** to remove it.

7 Click **Debug**  **Continue** to play the macro through, or click a Step option.



 **Related topics**

To create and edit variables in your macro,

 **1** Click **File**  **Debug**

 **Play.**

- 2** Specify the macro you want to debug.
- 3** Select the types of variables you want to display.
- 4** Select a variable, or double-click an array, then select a variable within it.
- 5** Edit the contents of the variable, then click **Update**.

 **6** Click **Variables**  **New**, type a name for the new variable, then click a variable type to add a variable to the macro.

 **7** Click **Variables**  **Delete** to remove a variable.

Tips

- ◆ The Variables list displays all of the variables defined at the current step in the macro. As you click the entries in the label, function, or procedure list, the variables in the list change, displaying the local variables defined in each step.
- ◆ Updating the value of a variable that is an alias for another variable changes the original value, and all other aliases will reflect the new value.

 [Related topics](#)


To debug macros,

 **1** Click **File**  **Debug**

 **Play.**

2 Specify the macro you want to debug.

3 Select the types of variables you want to display.

4 Click **Debug**  **Continue** to play the macro through, or click a Step option.

5 Correct the errors using the macro editor you selected and the Contents text box for variable values.


Tips

- ◆ Press **Shift+F1**, then click a control for specific information.

  **Related topics**

To generate a listing file for debugging macros,

- 1 Click **Tools**  **Settings**, then click the **Compile** tab.
- 2 Select **Generate listing file**, then click **OK**.

- 3 Click **File**  **Debug**

- 4  **Compile.**

- 4 Select the macro you want to compile, then click **Open**.


Tips


- ♦ The list file has the same name as the original macro, with a .WCL extension. Use any ASCII text editor to display the list file, so you can see which lines in your macro correspond to which line numbers.



[Related topics](#)

To search macro source code,

 **1** Click **Edit** **Find Line Number**, then specify the line number you want to find.

 **2** Click **Edit** **Find Text**, then specify the text you want to find.

3 Click **Find Next Breakpoint** or **Find Previous Breakpoint** to locate the breakpoints in the macro.

Tips

- You can search for text, line numbers, and breakpoints in the macro source code.
- Once you have specified the search text, you can click **Find Next** or **Find Previous** to find other occurrences.



[Related topics](#)

To select debugging options,



- 1 Click **Tools** **Settings**, then click the **Debug** tab.
- 2 Select the options you want to use for invoking the Debugger.
- 3 Select the options you want to use for recording debugging information during compiling.
- 4 Set your animation options.

Tips

- ◆ Animation allows you to step through macros line by line, automatically bringing up the debugging dialog box at each step so you can check the variables and other calls. You can specify how long the Debugger will pause before moving to the next step. Clicking inside the Debugger halts the animation, as do any breakpoints you have specified. Click **Debug** { bmc stprtictn.bmp } **Animate** to resume automatic play.



[Related topics](#)

About the Debugger Interface

The Debugger is designed to help you find and correct errors and other problems in your macros. The Debugger dialog consists of four main areas: The last three are lists separated by split bars you can use to adjust the size of the individual lists in relation to the others. Each of the three lists supports a context-sensitive (right mouse) menu containing items relevant to that list. The items available on these menus are also available from the toolbar and the main menu bar of the Debugger dialog.

Tips

- The splitbars between the Source List, Call History and Variable List windows may receive focus and when it has focus, its position can be changed using the arrow keys on the keyboard. The arrow keys will move the splitbar by 1 pixel. Holding the SHIFT key with the arrow keys will move the splitbar by 5 pixels. Holding down the CONTROL key will move the splitbar by 10 pixels.

The Debugger State Message Line

The Debugger state message line indicates whether the Debugger (or the macro) is active, and the reason the Debugger is active (such as a breakpoint on a statement start, error condition, and so on). While the Debugger interface is active, execution of the macro is suspended. This means that you cannot interact with prompts, message boxes, or dialogs that are being displayed by the macro. When the macro is playing, the state message line reads "Macro is running." You can use some debugger features while the macro runs (for example, setting breakpoints). All commands that would access information about the macro state, however, are disabled while the macro is running and the Debugger is inactive.

The Toolbar

You can display or hide the toolbar in the Macro Debugger. Displaying the toolbar is helpful, giving you instant access to a wide range of features. You can also customize the toolbar by double-clicking a blank space on the toolbar to display the Customize Toolbar dialog box. From this dialog, you can remove, add, or reorder the toolbar buttons. The same toolbar configuration is used when debugging all macros.

Since there is a limit on the number of variables that are initially displayed in the Macro Debugger, click

The Macro Source List

The macro source listing displays the source of the macro being debugged (taken from the compiler listing file). It displays a red arrow to the left of the line the macro will execute next. The left margin also contains an indicator showing which statements have breakpoints, and whether those breakpoints are enabled or disabled.

The macro source list also displays a floating tip about the variable, label, token, or command you place your mouse pointer over, showing details about the item. If the item is a variable that has not yet been defined, or a label defined in a "use" file that has not yet been loaded, then the Debugger may be unable to identify it and will display "??".

You can place a breakpoint on any macro source line containing macro statements by double clicking the source line. Once defined, a breakpoint may be enabled or disabled from the source listing as well. You can also display



the source of another macro file (such as a use file) in the macro source area by clicking **File Open**. The last 9 accessed macro files are listed on the File menu.

The Call History List

The call history list area lists the user-defined functions/procedures and labels that have been called, in reverse order. The current location is listed at the top. The name of the function/procedure or label displays along with the line number where execution within that function/procedure or label was interrupted, and the file that the function/procedure or label is contained in.

When you select entries in the call history list, the macro source for that macro is displayed in the source listing area, and the associated line is highlighted and indicated by a green triangle in the left margin (unless the top entry is selected, in which case the red arrow is displayed). The variables accessible to the macro at that point are then listed in the variables area below.

The Macro Variable List

The variables area displays the list of variables accessible to the macro at the indicated location (shown in the Call History List). Also displayed are the variable's pool type (Local, Global or Persistent), the type of value the variable contains, and the current value of the variable. You can restrict the list of variables to any combination of the Local, Global, or Persistent pool variable types. Even though multiple variables with the same name in different pools may appear in the list, only the most locally scoped variable with that name is accessible to the macro as it executes (e.g. If there is both a Local and a Global variable named B, only the Local B may be accessed by the macro. Once a variable has been declared with the DECLARE, LOCAL, GLOBAL, or PERSIST

statements in a macro, the variable will show up in this list even though its contents may be undefined.

Array variables are displayed in this list with their declared dimensions and a contents type of Array. You can expand array variables to display the individual array elements in the list, or collapse them to hide the individual elements. This lets you examine the individual elements of the array as normal variables. If a variable is an address (alias) parameter to a user-defined function/procedure, its contents type is displayed as Alias, and it may be expanded and contracted like an array to show the actual variable that it is mapped to. If an alias variable is mapped to a Global or Persistent variable, then the variable pool type is displayed appropriately. If it is a Local variable, then the pool type is displayed as "Local to Caller," to distinguish it from a variable that is local to the current function/procedure.

The list of variables may be sorted by any of the columns by clicking the column heading. You can toggle the sort order from ascending to descending by clicking the same heading a second time. If the variable list is sorted by variable name or by pool, then expanded array elements are kept with their corresponding array. Sorting by the other columns may cause elements of an array to become separated from each other, depending on the contents of the array element. The current sort column and sort order are indicated by a ">" or "<" symbol before the column heading name.

You can create new variables in any variable pool, and you can also create an array variable by specifying the dimensions of the array after its name. Variables are created with undefined contents; you supply the value at any time. You may also reset or discard a variable by pressing the Delete key. You should consider carefully before discarding a variable, since the macro may rely on the variable being defined. Discarding an array variable (not an array element), will first reset the contents of all of its array elements.

You can change the contents of a variable (or array element) by double clicking the variable or clicking the contents column for that variable. Update the contents of the variable by clicking outside the edit field, or pressing the Return key. The changes are canceled by pressing the Esc key. You cannot change the contents of an alias variable, but you can change the variable mapped to the alias. If you change the contents of an array, the new value is placed in all of its array elements. Changing the contents of an array element changes that element only, and does not affect the contents of any other array elements.

If you are interested only in certain macro variables, you can add those variables to the variable watch list. Displaying only a few variables makes it easier for you to see only the information you are most interested in. Only entire arrays or non-array variables may be watched; individual array elements cannot be watched separate from their corresponding parent array. When the variable watch list is displayed, it replaces the normal variables list.

Displaying Variables At each breakpoint, the Macro Debugger displays the contents of the current variables in the variable list, including the values of array variables, where the contents of all the elements are displayed in an abbreviated list. But when an array has a large number of elements, the display of all elements is time consuming and of little practical value, so only the value of the first 25 elements are displayed. However, the values of all array elements are still visible by expanding the array.

Expanding an Array in the Variable List Expanding the contents of an array in the Macro Debugger can consume a great deal of time and memory. To speed up the process, only the first 100 elements of an array are initially displayed. When the array is expanded, a warning prompt is displayed, asking if all elements should be expanded, or just the first 100 elements. An additional 101st element is also displayed, representing the remaining elements in the array, without their values. The remaining elements can be displayed by clicking the [...] icon next to the 101st element, or by collapsing the array and expanding it again, which prompts the same warning, allowing the user to select the option that expands all elements.

Refreshing the Variable List Whenever the Macro Debugger stops at a breakpoint, the contents of all variables are refreshed and displayed. When a macro contains a large number of variables, this process can be quite lengthy, so the Debugger can be configured to display on the first few variables, which speeds up the time it takes to refresh the list. The Debugger will initially display the first 100 variables, but at any time, the contents of all variables can be displayed by selecting the menu item "Show All Variables".



About Debugging Macros



About Information Windows in the Debugger



To choose debugging options for macros



To create new variables



To delete a variable while debugging a macro



To debug macros



To edit macros



To generate a listing file for debugging macros



To select debugging options



To watch certain variables



Related topics

To watch certain variables,

- 1 Right-click the variable in the macro variable list.
- 2 Click **Watch**.



- 3 Right-click the macro variable list, then click **View**  **Watch** to display only the variables you have marked as watched in the list.

Tips

- Once a variable is added to the variable watch list, that variable name is displayed in the list until you remove it from the list, even if the actual variable ceases to exist (for example, if it is discarded in the macro). If the variable ceases to exist, the pool type of the watch variable will display "out of scope."



[Related topics](#)

About Discard Variable

Use Discard Variable to remove a variable while debugging a macro. You can delete the variable from the Variables list box, or you can remove the contents of the variable but leave it in the Variables list box.



To delete a variable while debugging a macro



Related topics


To delete a variable while debugging a macro,

 **1** Click **File** **Debug**

 **Play.**

2 Step through the variable until the variable you want to discard appears in the Variables list box.

3 Click the variable name.

 **4** Click **Variables** **Delete.**


5 Click the discard option you want.

 **Related topics**

About Editing Macros

Use the Editor you specified in the Settings dialog box to edit existing macros. You can use any ASCII text editor to edit your macros, but some editors have special features. When you use Notepad, you can also create new macros by typing a new filename. When you use WordPerfect to edit your macros, you can use the PerfectScript Command Inserter from the Macro Property Bar to insert macro commands into your macros or to edit existing ones.



For information about controls on the dialog box, click  , then click a control.



To edit macros




To generate a listing file for debugging macros



Related topics

To edit macros,



- 1 Click **File**  **Edit**.
- 2 Specify the name of the macro you want to edit, then click **Edit**.
- 3 If necessary, click **Convert** to convert the macro for editing.
- 4 Make the changes you want to the macro, then exit the macro editor.

Tips

- ◆ Before you can edit a macro, you need to specify a macro editor in the Settings dialog box.
- ◆ You can also edit macros by selecting a macro file name in the Explorer, then clicking **Edit** from the QuickMenu.



[Related topics](#)

About Event Log

The Event Log displays any event that occurs during debugging. These messages include standard messages ("Debugger event logging enabled") and custom messages for which you supply the comment.

All breakpoints allow a message to be logged to the Debugger event log. When the breakpoint triggers, this message is written to the Debugger event log (if the event log is enabled). A breakpoint may be set up to log a message, to cause a break in the macro, or both. In the left margin of the breakpoint list, a hand symbol is displayed. A yellow hand indicates a breakpoint that will cause a break in the macro (and may also log an event message), while a blue hand indicates a breakpoint that will log an event message and not cause a break in the macro.



To enable event logging, click **Debug**  **Breakpoints**, then click the **Actions** tab. You can log standard and custom messages, which will then appear in the Event Log.



To make a debugger log file




Related topics

To make a debugger log file,

1 Click Debug  Breakpoints

 Event Log.

2 Select the types of events you want to enter into the log file, then click Close

3 Start the Debugger, then click View  Event Log.

4 Select Logging enabled.

5 Click Save, then specify a path and filename for your event log.

Tips

- ◆ Your event log is a text file with a default .log extension.



Related topics

About Execute Token

When the Macro Debugger is active and the macro is suspended (the macro has stopped at a breakpoint), the Debugger allows you to execute any PerfectScript command (token) in a very localized temporary environment. It displays the PerfectScript commands along with their parameters and types, and lets you select a command and specify a value for each parameter. When you execute the command, the return value is displayed, and if you specified a variable name for the return value, then the return value will also be assigned to that variable.

Be careful when using Execute Token, since some PerfectScript commands may cause the internal state of the running macro to change and may cause errors to occur later in the macro. Most of these commands have been eliminated from the command list, and cannot be selected.

Values cannot be assigned to variables that have the same name as command token names.

3rd Party Tokens PerfectScript supports 3rd party token handler DLL's. All tokens (not just its own) are passed to the ValidateToken entry point. If the 3rd party DLL does not accept or approve the token, it can cause PerfectScript to abort the macro. PerfectScript's own tokens are passed to the HandleToken entry point in the 3rd party DLL's.



To execute a macro token while debugging



Related topics

To execute a macro token while debugging,



- 1 Click **View** **Execute Token**.
- 2 Select the PerfectScript token you want to execute, then specify any parameter and return values you want to use.
- 3 Click **Execute** to perform the command and display any return value.

Tips

- Be careful when using Execute Token, since some PerfectScript commands may cause the internal state of the running macro to change and may cause errors to occur later in the macro. Most of these commands have been eliminated from the command list, and cannot be selected.
- A keyboard accelerator of ALT+0 has been added for the Execute Token dialog.



Related topics

About Information Windows in the Debugger

The Debugger includes several separate informational windows you can open to display various types of information about the current state of the macro. These windows are modeless dialogs and are refreshed whenever the Debugger becomes active. Most of the windows display information that is specific to the current execution point in the macro, but by selecting a different entry in the Call History list of the Debugger, you can display information specific to the selected entry.

The info windows in the Debugger have context menus containing options to navigate from that window to the other info windows the main Debugger window and to the matching macro source line, if any. In any of these windows where labels, line numbers, or filenames are displayed, double-clicking that item will locate that position in the macro and display it in the macro source list. These items are indicated by a gray arrow in the left margin. These are the information windows available:

Label Table: A list of all the labels defined at the execution point selected in the Call History list. For each label, the label name, type, line number, and file name are displayed. The label type is either local or global. Local labels are defined by the Label statement in a macro, and are visible only within the procedure/function where they are defined. Global labels are user-defined procedures and functions, and are visible anywhere in a macro file, as well as in other macro files that have a Use statement of the file containing the procedure/function. The name of the file where the label is defined is displayed, as well as the line number of the source line where the label or procedure/function is defined. Double-click any of these labels to display the macro file in the macro source list window, and highlight the source line containing the label definition.

Use File Table: A list of all Use files referenced in Use statements by the macro file selected in the Call History list. If the labels for that Use file have been loaded (this happens the first time a procedure/function is called from the Use file), then the Loaded column shows True. Double-click any Use file in this list to load that file into the macro source list window.

Product Table: A list of all the applications/products that have commands in the macro file selected in the Call History list. If an Application statement for an application is in a macro, but the macro does not actually contain any commands for that application, that application will not be displayed in this list. For each application/product listed, the version number of the PID file (product interface description file) that was used when this macro was compiled is displayed. This version number is used to determine if a compiled macro has become out of date when a new version of an application is installed, and the macro is played.

Dialog List: A list of all user-created macro dialogs that are currently defined or that exist in the prefix packet of the macro file selected in the Call History list. For each dialog, the name, state, type, callback label, position/size, and styles are displayed. There are 2 types of dialogs: Text dialogs, which are defined using DialogDefine, and DialogAdd statements in a macro; and Binary dialogs, which are created by the Macro Dialog Editor, and are stored in the prefix packet area of a macro file. The states of a dialog are:

- Defined: Text dialogs only - the dialog has been defined by a DialogDefine statement, but the dialog has not been loaded or shown yet.
- In Prefix: Binary dialogs only - the dialog was found in the prefix packet of the current macro file, but hasn't been loaded or shown yet.
- Loaded: Text and Binary dialogs - the dialog has been loaded by a DialogLoad statement, or by a Region command.
- Showing: Text and Binary dialogs - the dialog is currently showing by a DialogShow statement.

If the dialog is currently showing, and a callback label was specified, then the Callback label is displayed. Double-click this dialog to display the macro file where the callback label is defined in the macro source list window, with the source line containing the label definition highlighted. The position and size of the dialog when it was defined is shown, but this does not display the current position or size of the dialog. The styles associated with this dialog in the DialogDefine statement or in the Macro Dialog Editor are also shown. In the lower half of this window, the list of controls defined for the selected dialog are displayed. For each control, its order, name, type, position/size, associated variable, styles, and data are shown. As with the dialog position and size, the position and size of the control that is displayed is the position and size when the control was defined, not its current position and size.

Condition Handlers List: A list of all condition handlers defined for the execution point selected in the Call History list. For each condition handler, an action and data are displayed. The action column tells you whether the condition will cause the macro to abort or quit, or whether the condition will cause a label to be called or jumped to, and the appropriate label. If the handler has been disabled, the action will say Ignore, indicating that the abort, call, or jump will be ignored. If there is a label associated with a condition handler, double-click the item to display the macro file where the label is defined in the macro source list window, and to highlight the source line containing the label definition. The standard condition handlers, such as Error, Cancel, and NotFound are displayed in this list, as well as handlers for callbacks such as OnDDEAdvise and callback dialogs.

Macro Info List: A list of all the data that can be obtained from the MacroInfo command in a macro for the execution point selected in the Call History list. Some of these items can have labels, line numbers, or filenames associated with them. Double-click these items to display the macro file in the macro source list window and to highlight the source line containing the label definition or line number.

Callback Queue: A list of all items in the callback queue which indicates which callback are currently active and

which are pending. It is possible for multiple callbacks to be active at the same time. The callback queue contains entries for callback dialogs and for OnDDEAdvise notifications. The label that will be called by this callback is specified. Double-click this line to display the macro file in the macro source list window and to highlight the source line containing the label definition. The status column indicates whether this callback is for notification only, or whether the callback can affect the action performed by the macro system when the callback is complete. Dialog callbacks are always for notification only. The contents of the callback data array is also displayed, and an interpretation of the specific array elements is also displayed where possible. Callback can be removed from the Callback Queue list. To remove a callback, press the DELETE key to delete the currently selected callback entry from the Queue. A warning displays to confirm the action, since deleting a callback can dramatically alter the behavior of the macro.

Macro Header: A window that displays the macro object header information for this macro file, including the version number of the macro system used to compile this macro file. This window is a modal dialog.

Tip

The following accelerator keys are available in the Debugger info windows:

Space jump to the associated macro source line (if any).

DEL/Backspace delete the current callback queue entry.

ALT 0/ALT 9 Display one of the Debugger info windows.




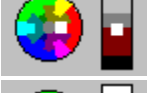






CTRL Home Set focus to main Debugger window.

CTRL F6 Set focus to next Debugger window.

CTRL Down Arrow Set focus to next Debugger window.

SHIFT|CTRL F6 Set focus to previous Debugger info window.

CTRL Up Arrow Set focus to previous Debugger info window.

	<u>To display a list of all the applications/products that have commands in the macro file</u>
	<u>To display a list of all condition handlers defined</u>
	<u>To display a list of all the data that can be obtained from the MacroInfo command</u>
	<u>To display a list of all the labels defined at the execution point</u>
	<u>To display a list of all pending items in the callback queue</u>
	<u>To display a list of all Use files referenced in Use statements by the macro file</u>
	<u>To display a list of all user-created macro dialogs</u>
	<u>To display the macro object header information for this macro file</u>
	<u>To display all variables in the Macro Debugger</u>
	<u>To close all of the Debugger information windows</u>



To locate the current line of code where a macro has been interrupted



Related topics

To display all variables in the Macro Debugger,

1 While debugging a macro, click **Variables|View|All..**



Related topics

To close all of the Debugger information windows,

1 While debugging a macro, click **View|Close All..**



Related topics

To locate the current line of code where a macro has been interrupted,

1 While debugging a macro, click **Edit|Find Current Line**.

Note

- The entire line of code in the source window is highlighted.



Related topics

To display a list of all the labels defined at the execution point,



1 While debugging a macro, click **View** **Label Table**.

Tips

- Double-click any of these labels to display the macro file in the macro source list window, and highlight the source line containing the label definition.



Related topics

To display a list of all Use files referenced in Use statements by the macro file,



1 While debugging a macro, click **View**  **Use File Table**.

Tips

- Double-click any Use file in this list to load that file into the macro source list window.



Related topics

To display a list of all the applications/products that have commands in the macro file,



1 While debugging a macro, click **View** **Product Table**.

Tips

- For each application/product listed, the version number of the PID file (product interface description file) that was used when this macro was compiled is displayed. This version number is used to determine if a compiled macro has become out of date when a new version of an application is installed and the macro is played.



Related topics

To display a list of all user-created macro dialogs,



1 While debugging a macro, click **View** **Dialog List**.

Tips

- There are 2 types of dialogs: Text dialogs, which are defined using DialogDefine and DialogAdd statements in a macro; and Binary dialogs, which are created by the Macro Dialog Editor, and are stored in the prefix packet area of a macro file.
- Double-click this dialog to display the macro file where the callback label is defined in the macro source list window, with the source line containing the label definition highlighted.



Related topics

To display a list of all condition handlers defined,



1 While debugging a macro, click **View** **Condition Handlers**.

Tips

- If there is a label associated with a condition handler, double-click the item to display the macro file where the label is defined in the macro source list window, and to highlight the source line containing the label definition.
- The standard condition handlers, such as Error, Cancel and NotFound are displayed in this list, as well as handlers for callbacks such as OnDDEAdvise and callback dialogs.



[Related topics](#)

To display a list of all the data that can be obtained from the MacroInfo command,



1 While debugging a macro, click **View** **Macro Info List**.

Tips

- Double-click these items to display the macro file in the macro source list window and to highlight the source line containing the label definition or line number.



Related topics

To display a list of all pending items in the callback queue,



1 While debugging a macro, click **View** **Callback Queue**.

Tips

- Double-click a line to display the macro file in the macro source list window and to highlight the source line containing the label definition.



Related topics

To display the macro object header information for this macro file,



1 While debugging a macro, click **View** **Macro Header**.

Note

- ◆ The Macro Header info window displays whether or not the macro is protected and if it contains its source code.



Related topics

About Moving Through Macros While Debugging

When the Debugger stops on a macro statement, it stops before the indicated statement has been executed. To continue execution of the macro, a number of different options are available.

Continue: Macro execution is continued to the next breakpoint.

Step Into: Execute the next single statement. If the next statement is a label or routine call, then execution will step into the specified label or routine, even if that label or routine is in another macro file (such as a use file).

Step Over: Execute the call of the label or routine without stopping until it has completed. This will stop the macro at the next statement in the current label or routine.

Step Out: If you have entered a label or routine, execute until the next return is encountered.

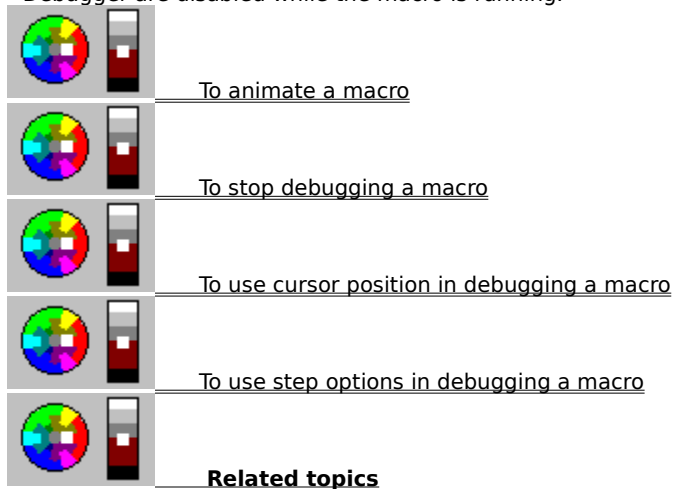
Run to Cursor: Continue execution down to the line under the mouse cursor in the macro source list.

Skip to Cursor: If you need to skip a series of statements without executing them, or if some statements need to be repeated, this sets the next statement that will be executed by the Debugger, without executing any statements between the current point and the new line. This option should be used with extreme caution, since skipping to a line that is not within the same label or routine could cause the internal macro execution state to become invalid, which will almost certainly result in execution failure.

Animate: When this mode is turned on, then the next Continue or Step command is repeated until you stop it, or until the macro ends. Between each command, the Debugger is displayed for the amount of time you specify in the Settings dialog box.

Stop: Stops the macro and the debugging. When you stop a macro this way, all defined breakpoints, watchlist variables, and opened macro files are stored in a debugger configuration file specific to the macro being debugged, and are saved in a file with a ".dbg" file name extension in the same directory as the macro. This configuration file is reloaded the next time that macro is debugged.

Break: While the macro is executing statements, the Debugger is mostly inactive, and the Debugger state message line displays the text "Macro is running." However, the Debugger is not completely disabled. You can interrupt the macro by clicking **Break**. This will cause the Debugger to become active, as if there were a breakpoint at the point where the macro is currently executing. The breakpoint commands are also available while the macro is running, so that you can add and remove breakpoints. Most of the other features of the Debugger are disabled while the macro is running.



To animate a macro,

 **1** Click **Edit** **Settings**

 **Debug.**

2 Select the type of animation you want to use (Step Into or Step Over).

3 Specify the number of seconds the macro should display after executing each step, then click **OK**.

 **4** Click **Debug** **Animate** to begin playing the macro.

Tips

- Animation allows you to step through macros line by line, automatically bringing up the debugging dialog box at each step so you can check the variables and other calls. You can specify how long the Debugger will pause before moving to the next step. Clicking inside the Debugger halts the animation, as do any breakpoints you

 have specified. Click **Debug** **Animate** to resume automatic play.

- Step Into executes the next single statement. If the next statement is a label or routine call, then execution will step into the specified label or routine, even if that label or routine is in another macro file (such as a use file).
- Step Over executes the call of the label or routine without stopping until it has completed. This will stop the macro at the next statement in the current label or routine.

 **Related topics**

To stop debugging a macro,



- 1 While you are debugging a macro, click **Debug**  **Stop Debugging** to quit the macro and close the Debugger.

Tips

- When the macro is stopped and the Debugger closes, all defined breakpoints, watchlist variables, and opened macro files are stored in a debugger configuration file specific to the macro being debugged, and are saved in a file with a ".dbg" file name extension in the same directory as the macro. This configuration file is reloaded the next time you debug the macro.



Related topics

To use cursor position in debugging a macro,

1 While debugging a macro, position the cursor by clicking in the macro source list.



2 Click **Debug** **Run to Cursor** to continue execution down to the line under the mouse cursor in the macro source list.



3 Click **Debug** **Skip to Cursor** to set the next statement that will be executed by the Debugger, without executing any statements between the current point and the new line.

Tips

- You can use Skip to Cursor if you need to skip a series of statements without executing them, or if some statements need to be repeated. This option should be used with extreme caution, since skipping to a line that is not within the same label or routine could cause the internal macro execution state to become invalid, which will almost certainly result in execution failure.



[Related topics](#)

To use step options while debugging a macro,



- 1 While debugging a macro, click **Debug Step Into** to execute the next single statement. If the next statement is a label or routine call, then execution will step into the specified label or routine, even if that label or routine is in another macro file (such as a use file).



- 2 Click **Debug Step Over** to execute the call of the label or routine without stopping until it has completed. This will stop the macro at the next statement in the current label or routine.



- 3 Click **Debug Step Out** to execute the macro until the next return is encountered, if you have entered a label or routine.



[Related topics](#)

About the PerfectScript Language

The PerfectScript language is a command-based language. This means that the application records the results of keystrokes or mouse selections instead of recording the keystrokes themselves. For example, instead of recording each keystroke involved in changing the top margin of your document to 2, the application simply records the command `MarginTop(2.0)`. These commands are called product commands.

The PerfectScript language also includes programming commands. These commands let you create macros that can respond to user input.



Example of DialogShow and DialogDismiss




Related topics

About Playing Macros

Use macros to carry out procedures inside Corel suite applications. You can play, pause, resume, and stop macros. When you play a macro, PerfectScript performs the operations and uses the features you specified in the macro.



For information about controls on the dialog box, click , then click a control.



To pause and resume macros



To play macros



To stop macros



Related topics

To play macros,



- 1 Click **File** **Play**.
- 2 Select the macro you want to play, then click **Play**.

Tips

- ◆ You can also play macros by selecting a macro file name in the Explorer, then clicking **Play** from the QuickMenu.



[Related topics](#)

To pause and resume macros,

- 1 While a macro is playing, click **PerfectScript** on the **Start** bar.
- 2 Select the macro you want to pause, then click the **Pause** button on the toolbar.
- 3 When you want the macro to continue playing, click the **Pause** button again.

Tips

- You can also right-click a macro, then click **Pause** from the QuickMenu.



[Related topics](#)

To stop macros,

- 1 While a macro is playing, click **PerfectScript** on the **Start** bar.
- 2 Select the macro you want to stop, then click the **Stop** button on the toolbar.

Tips

- ◆ You can also right-click the macro, then click **Stop** from the QuickMenu.




Related topics

About Recording Macros

Use Record, Pause Record, and Stop Record while creating keyboard macros. When you record, PerfectScript keeps track of the actions you perform in an application, such as typing text or saving a file.



For information about controls on the dialog box, click , then click a control.



To record macros



Related topics

To record macros,



- 1 Click **File** **Record**.
- 2 Type a name for the macro, then click **Record**.
- 3 Switch to a WordPerfect suite application or another macro-enabled program, then use it to record keyboard actions in the macro.



- 4 When you have finished recording the macro, click **File** **Stop Record**.

Tips

- ◆ Press **Ctrl+x** or **Ctrl+Shift+x** (where x represents a character a-z or 0-9) to assign your macro to a keystroke combination.
- ◆ If you need to stop recording temporarily to locate a feature or experiment with the effect of a feature before



you record the command, click **File** **Pause Record**, then click **Pause Record** again to continue.

- ◆ The suite application you are using automatically inserts the Application command when you record a macro. This command tells PerfectScript that, unless otherwise specified, commands with that prefix should be sent to that application and not to another.




Related topics

About Settings

Use Settings to select default settings for recording, editing, debugging, compiling, playing and storing macros.



For information about controls on the dialog box, click , then click a control.



To select a macro editor



To select default macro settings



To choose recording options for macros



To generate a listing file for debugging macros



To select debugging options



To select general macro settings



To select macro recording defaults



Related topics

To choose a macro editor,



- 1 Click **Tools** **Settings**, then click the **Edit** tab.
- 2 Specify the path and filename of the macro editor you want to use.
- 3 Specify the file format of the editor you selected.

Tips

- ◆ You must select a macro editor before you can edit macros from PerfectScript, and the file format must match the editor you select.



Related topics

To choose default macro settings,



- 1 Click Tools Settings.
- 2 To select the default macro folder, set Display icons in system tray, and reset defaults, click the General tab.
- 3 To show progress during compiling and to select other compiling options, click the Compile tab. The "Warn when using unsupported features" option defaults to True.
- 4 To select debugging defaults, click the Debug tab.
- 5 To select a macro editor and file format, click the Edit tab.
- 6 To select play options for macros, click the Play tab.
- 7 To select script language, file format, numbers of parameters, prefixes, named parameters and maximum line length, click the Record tab.
- 8 To set toolbar macro button options, click the Toolbar tab.

Tips

- The PerfectScript toolbar displays three buttons that can be configured to play any macros that you choose. To configure the buttons, select **Tools, Settings**, then click the **Toolbar** tab. Choose the macros that you want associated with each button.



[Related topics](#)

To select macro play settings,



- 1 Click **Tools** **Settings**, then click the **Play** tab.
- 2 Select the script language and file recording format you want to use.
- 3 Select the number of parameters you want per line.
- 4 Specify the maximum line length.
- 5 Select **Named parameters required** to force all parameters to have names.
- 6 Select **Record product prefixes on all commands** if you want to use product prefixes in macros.

Tips

- ◆ The file format you select in step 2 should be compatible with the macro editor you select on the Edit tab.



Related topics

To select general macro settings,



- 1 Click **Tools** **Settings**, then click the **General** tab.
- 2 Specify a default macro folder.
- 3 Select **Use Enhanced File Dialogs** if you want to view detailed dialog information.
- 4 Select **Display Icons in System Tray** if you want to enable macro icons in the Windows system tray.

Tips

- ◆ Click **Reset All to Defaults** to return all settings to their original state.



[Related topics](#)

To select macro recording settings,



- 1 Click **Tools** **Settings**, then click the **Record** tab.
- 2 Select the script and file formats in which you want to save your macro.
- 3 Click **One** or **Multiple** from the Parameters per line drop-down list to set the number of parameters for each line in your macro.
- 4 Specify a maximum line length for your macro text.
- 5 Select **Named parameters required** to use only named parameters in your macros.
- 6 Select **Record product prefixes on all commands** if you want to insert product prefixes in your macros.

Tips

- ◆ The file format you select in step 2 should be compatible with the macro editor you select on the Edit tab.



[Related topics](#)

Tips for Debugging Macros

The actual techniques you use to debug a macro depend on the type of failure that is occurring. To begin debugging a macro, first compile the macro for debugging. It is also a good idea to use the Debugger to compile any other macro file that the original macro Uses or Runs or Chains to. Compiling a macro for debugging includes necessary information in the macro, and generates a compiler listing file the Debugger can use to display the macro's source code.

Debugging Macros that Stop at Error Messages

- 1 One of the easiest problems to correct with the Debugger is when a macro terminates due to an error. To use the Debugger to discover the cause of the error, play the macro as usual. When it stops, press the Debug button on the error message box. This will bring up the Macro Debugger, and it will load the macro and the compiler listing file. The Debugger will display the problem line in the macro source list. It will also display, in the call history list, a reverse-ordered list of what labels and procedures/functions were called in order to get to that point. The contents of all the macro variables will also be displayed.
- 2 At this point, examine the source line to determine what the macro was doing when the error occurred. Check the contents of the variables that are being used at the current line to see if they contain the proper values. Incorrect values for some of the variables' contents can cause macro errors.
- 3 If all the variables at this point in the macro appear to have the right values, select a previous line in the call history list, and examine the source line and the contents of the variables at that point. If you find an improper variable value, examine the source code leading to the problem area. Locate a spot where the variable could have been changed, and double-click the line to set a line number breakpoint at that spot.



- 4 Now stop the Debugger by clicking **Debug**  **Stop Debugging**. You cannot continue executing the macro, since an error has occurred and the macro cannot continue.



- 5 Restart the macro by clicking **File**  **Debug**



Play in PerfectScript. This displays the Debugger right before the macro starts.

- 6 Since you set a line number breakpoint, click **Continue** and run the macro until it encounters the line number containing the breakpoint.

- 7 When the macro reaches the line with the breakpoint, the Debugger displays. Examine the variable contents to verify that the macro is displaying the variable values you expected. If not, you may need to specify a breakpoint at an earlier location and restart the macro. You can also skip to the earlier location by clicking a



line in the macro source, then clicking **Debug**  **Skip to Cursor**.

- 8 If everything appears to be working normally, click **Step Into** to execute the macro one statement at a time. At each step, examine the variables between each statement until something unexpected happens. For example, the contents of a variable may change unexpectedly, or the macro may call the wrong label. At this point, you have probably come to the line where the variable's value causes the error.

- 9 If it appears that the error occurs because a variable has the wrong contents, set a "variable assign"



breakpoint for that variable by clicking **Debug**  **Breakpoints**



Edit, then selecting **Variable Assign**. When the contents of that variable change, a breakpoint occurs and the Debugger displays. If the variable is changed too often, and too many breakpoints are occurring on the variable, disable the Variable Assign breakpoint and set a line-number breakpoint after most of the variable assignments are done, but before the wrong contents are assigned. If the problem in the macro appears to be associated with calling a certain product token, set a Product Token Call breakpoint for that product token, and/or for a related product token that may not be setting the application in the proper state. If the problem is that the macro is calling a specific macro label or procedure/function too often, set a Label Call breakpoint for that label to see when and where it is being called.

Debugging Callbacks

Debugging dialog callbacks is one of the most challenging types of debugging. Many problems with dialog callbacks occur because the callback does not look at the callback data closely enough to determine if the exact conditions have occurred before performing the macro's action; so the action is performed at the wrong time or too many times.

A dialog callback label is called often, and most of the callbacks are usually not associated with the event of interest. As soon as a callback dialog is shown, at least 2 or 3 callback events occur. These events are associated with the dialog being created, the controls being initialized, and with the dialog receiving the initial input focus.

One of the difficulties associated with debugging callbacks is that when the Debugger becomes active at a breakpoint, it gains focus. This means that the window that had focus while the macro ran lost focus, and therefore the callback dialog in the macro lost the focus as well. When a callback dialog loses or gains focus, it generates a callback event.

If you set a breakpoint in a dialog callback label, a lose-focus callback event could occur on the callback dialog when the Debugger displays. Then, when you continue macro execution in the Debugger, a gain focus callback event could occur. This could cause the breakpoint to stop the Debugger (because the callback label is called for the lose/gain focus of the dialog), which would in turn cause more lose/gain callback events, which could cause another breakpoint, which could cause more lose/gain focus callback events, and so on.

The macro interpreter and debugger attempt to minimize this cyclic occurrence by determining whether the dialog is losing or gaining focus because of the Macro Debugger. If so, the focus callback events are not generated. Unfortunately, this cannot always be reliably determined.

The best way to prevent error loops is to write the dialog callback label so that it carefully examines the callback data array to determine the type of the callback event, and have it respond only to the callback event types it is interested in. (Callbacks usually respond to events associated with manipulating a control, which are WM_COMMAND events with element [5] = 273). Then set a breakpoint on a line of code that does not respond to losing or gaining focus. In this way, the breakpoint does not occur when the dialog loses or gains focus, circumventing the error cycle.

If a callback event occurs for a callback label different from any callback label that is currently executing, then the callback even will cause the current callback code to be suspended, and the new callback code will be called. By restricting this to different callback labels than any currently active callback label, this ensures that a callback will complete before another callback for that same callback label is allowed to occur.



Related topics

Tips for Linking Dialogs to Macros

Use the DialogShow and DialogDismiss commands to use a dialog created in the PerfectScript Dialog Editor within a macro. These commands can be found in the PerfectScript Command Inserter under the type PerfectScript.

The DialogShow command uses three parameters: the dialog name, the parent window for the dialog, and a label that identifies a callback function. The first and second parameters are required, the third is optional. The DialogDismiss command requires two parameters: the name of the dialog and the name of the control used to dismiss the dialog. See the Example cited in the Related topicslist for syntax and further information.

Variables associated with controls work the same way as with the DialogDefine method. If a variable exists, its value is set into the controls when the dialog displays, and the value in the controls is set into the variables when the dialog is dismissed.

Before a you can use a region command to manipulate a dialog, that dialog must be loaded into memory. DialogShow loads the dialog into memory and displays it. (The DialogLoad command loads the dialog into memory but does not display it.) Because a dialog must be loaded before you can use it, be sure to use a DialogShow (or DialogLoad) command before you use any Region commands.




[Related topics](#)

Tips for Locating Applications

When you play a macro, PerfectScript checks the Registry for the path to the application's .EXE. If the location is not in the Registry, PerfectScript asks you to specify the location of the application's .EXE.

- 1 Select the macro you want to play, then click **Play**.
- 2 Specify the location of the .EXE file for the application.



For information about controls on the dialog box, click , then click a control.



[Related topics](#)

Tips for Macro Commands and Syntax

If you create macros by recording keystrokes and mouse clicks, the commands are automatically inserted in the correct format.

For macros you type from the keyboard to work properly, macro commands and their elements must be arranged in the correct order. This arrangement is called syntax. To be syntactically correct, each macro command must be spelled correctly and must include all of the required parameters and the necessary separators in the correct order.

Macro commands consist of three parts: a command name, parameters, and separators.

Command Name

The command name indicates which feature the command activates. Sometimes the name is all that is necessary to perform a complete action. For example, `FileOpenDlg()` is a complete macro command because WordPerfect does not need any additional information; the command name itself opens the File Open dialog box.

Example: `FileOpenDlg()`

Parameters

If the token or command needs more information than is provided by the command name alone, parameters are required. The command name represents the feature. Parameters represent aspects of the feature you can change, or selections you can make. For example, the Backup command requires one parameter that indicates whether you want Automatic Document Backup turned on or off. Parameters are always enclosed in parentheses.

Example: `Backup(On!)`

Separators

Some macro commands require several parameters. Parameters generally need to be placed in the correct order (except when you specify parameter names) and must be separated properly. Semicolons (;) are used to separate individual parameters, and an entire string of parameters must be enclosed in parentheses. Groups of repeating parameters are also enclosed in braces ({}).

For more information, see Expressions and Variables in the online macros manual. In addition, see the Macro Commands Index in the online macros manual for the required syntax and a description of each parameter for all PerfectScript macro commands.

Typing Macro Commands Yourself

Since you do not need to use a macro editor to create or edit WordPerfect macros, you can insert macro commands by typing them in a blank document. When you save a macro, remember to give it a `.WCM` extension. You can also type commands into a macro that is saved in a file or in a template.

If you want to improve the readability of a macro, you can format it so that it includes tabs, spaces, and even font or text appearance changes. Formatting the macro will not affect how it works. For example, WordPerfect records the following macro in this format:

```
PosDocBottom()  
Type("Sincerely")  
HardReturn()  
HardReturn()  
HardReturn()  
HardReturn()  
Type("Ms. Sharon Openshaw")  
HardReturn()  
Type("Vice President, Marketing")
```

However, if you type in the commands yourself or edit the existing macro, you can leave spaces between components as follows:

```
PosDocBottom()  
  
Type ("Sincerely")
```

HardReturn()

HardReturn()

HardReturn()

HardReturn()

Type ("Ms. Sharon Openshaw")

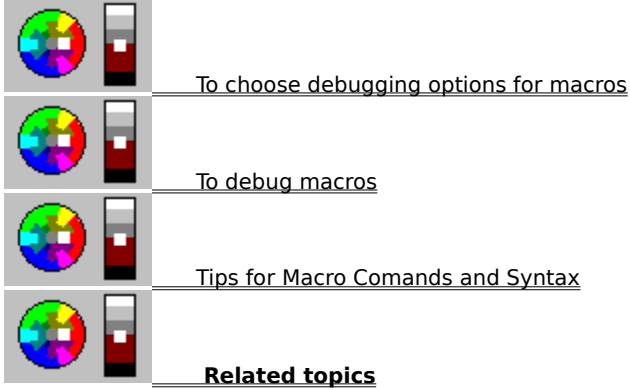
HardReturn()

Type ("Vice President, Marketing")

Troubleshooting Macro Problems

The compiler is a useful tool for troubleshooting syntax problems in your macros. If the compiler locates an error, the compiler displays a dialog box containing general information about the problem. The macro compiler, however, can only make a best guess as to what the macro is actually designed to accomplish. As a result, the information the compiler gives sometimes suggests, rather than specifically identifies, the problem.

Some Corel suite applications also have an online macros manual available on the CD. The online manual contains information about macro commands and their syntax, with additional instructions and examples for using macros.



Specify the justification (left, center, or right) for group box titles.

Specify the text associated with the edit box.

Put a title bar and caption on the dialog.

Put a system menu on the dialog.

Select the types of scrolling the edit box allows beyond its edges. Vertical scrolling allows text to scroll up and down. Horizontal scrolling allows text to scroll left and right.

Size the control to make it as large or small as the bitmap.

Size the bitmap to make it fit on the control.

Do not size either the control or the bitmap. If the bitmap is bigger than the control, the edges are cropped.

Select the state of the radio button when the dialog opens. Clear leaves the radio button unmarked and unactivated. Checked marks the radio button with a dark circle and leaves it activated.

Specify the text appearing on a push or pop-up button, or beside a radio button.

Select how text appears in the edit box (upper, lower, or mixed case).

Specify the text that appears in the title bar of the dialog. If you type a title longer than the title bar, it is cut off at the right edge.

Specify the text that appears next to the check box.

Create a check box that gives two possible states (on or off). When users click the check box, it automatically changes the state. No additional message is necessary in the macro. This is the most commonly used type.

Create a check box that gives two possible states (on or off). When users click the check box, it does not automatically reflect the change. You must send a message to the dialog within the macro to change the state of the check box.

Create a check box that gives three possible states (on, off, or indeterminate). When users click the check box, it does not automatically reflect the change. You must send a message to the dialog within the macro to change the state of the check box.

Create a check box that gives three possible states (on, off, or indeterminate). When users click the check box, it automatically changes the state. No additional change is necessary in the macro.

Specify the dialog class.

Allow users to enter text longer than the actual combo box without cutting it off.

Convert text typed in the control from the Windows character set to the OEM character set, then back to the Windows character set again. This ensures proper character conversion.

Create a combo box as a small, permanently open list with a thin, black border.

Create a combo box as a small list with a thin, black border when the user clicks the icon next to the edit control. A scroll bar appears if the number of items in the list exceeds the length of the list box.

Create a combo box as a static text field displaying the current selection instead of an edit control. Clicking the control displays the list. A scroll bar appears if the number of items in the list exceeds the length of the list box.

Accept or cancel changes the user made to the dialog box.

Allow users to enter numbers longer than the actual counter text box without cutting them off.

Automatically check entered numbers to ensure they fall within the acceptable range, and change unacceptable numbers to the nearest acceptable value when the user leaves the control.

Remove the incrementor/decrementor buttons.

Specify the minimum value possible for the counter.

Specify the maximum value possible for the counter.

Specify the increment the counter's incrementor/decrementor buttons use. For example, typing 5 causes each click on the Up arrow to increase the value in the text box by 5: 5, 10, 15, 20, and so on.

Specify the value the counter displays when the dialog opens.

Specify the unit the counter measures (i.e., centimeters, inches, points, and so on).

Specify the list of possible choices. Click Create/Edit List to add, replace, delete, and position items on the list, and to set the option selected when the dialog opens.

Automatically check entered dates to ensure they fall within the acceptable range, and keep the focus on the control until the user enters an admissible date and format.

Create a dialog that a user must dismiss to access features in the parent window. You can switch to another window using Alt+Tab.

Create a dialog that a user can leave open and still access features in the parent window.

Create a dialog that the user must dismiss before accessing any other functions.

Allow users to press Return inside the edit box to enter multiple lines.

Display characters as asterisks as the user types a password.

Display text the user cannot edit.

Allow text to wrap to the next line when it reaches the right edge of the text box.

Specify the sides of the edit box on which scroll bars appear. Use scroll bars with edit boxes to let the user see all the text entered.

Limit text entry to a single line in the edit box. Pressing Return moves the user to the next control.

Allow users to enter multiple lines of text in the edit box.

Limit text entry to the number of characters you specify. The system limit is 1 K.

Specify the path and filename of the bitmap image you want to use. Use the browse icon to select paths and filenames.

Display only folder and subfolder names in the browse list; all filenames are grayed.

Display folders, subfolders, and filenames in the browse list.

Allow users to enter invalid filenames without generating an error message.

Use with Path Only. Allow users to enter invalid paths and to create the folder or subfolder.

Specify the default folder and template for filename boxes. Click the folder icon to select folders.

Select a white, black, or gray fill and border for frames.

Remove the frame from around the dialog.

Put a thin black line around the dialog.

Put a Windows-standard dialog border around the dialog.

Put a thick border around the dialog with sizing areas on the corners and edges.

Select the type of frame you want to use. Using frames does not automatically group controls. Select Filled to fill the area inside the frame with a solid color. Select Frame to outline the area with a thin line.

Specify the Help file and Help key this dialog connects to. For example, you can tell this dialog to look for "MYHELP.HLP" and assign it a key included in the map and alias files for that Help system.

Select the buttons displayed on the dialog that appear when users click the folder icon. You must have these applications and options available in order for them to operate.

Display a button that opens QuickFinder.

Display a button that opens a dialog allowing users to change the appearance of the dialog.

Display a button that allows users to view files.

Display a button that allows users to copy, rename, delete, and otherwise manipulate files.

Specify the values for the color highlighted when the dialog opens. You can also click the color wheel and color bar to select colors.

Adjust the color (0=red, 240=blue, 360=green).

Adjust the amount of white and gray in a color.

Adjust the amount of color.

Specify the date displayed when the dialog opens. Click the calendar icon to select the date, or specify today's date.

Create a check box that is unmarked and unactivated when the dialog opens.

Create a check box that is marked with an X and activated when the dialog opens.

Create a check box that is filled with gray as the dialog opens. You must select Three-State or Auto Three-State from the type group box to activate this option.

Specify the justification (left, center, or right) for text inside edit boxes.

Display check boxes next to each item in a list box.

Allow users to select several separated items in the list box by pressing Ctrl while clicking.

Allow users to select contiguous items in a list box by dragging over them.

Precisely position a control in relation to the top-left corner of the dialog. The values you can use depend on the size of the dialog and are measured in dialog units. You can also select and drag controls to position them.

Record any events that occur during debugging.

These are the events that occurred during debugging, including the time and date.

Position the dialog in dialog units from the top-left corner of the parent window. Dialog units are based on the currently selected font for the dialog and on your screen resolution.

Position the dialog based on available screen space. For example, if you choose 50% for Left and 50% for Top, 50% of the available screen space will be on the left and 50% will be at the top of the dialog.

Specify the percent or number of dialog units the dialog is removed from the left edge of the window.

Specify the percent or number of dialog units the dialog is removed from the top edge of the window.

Specify the width of the dialog in dialog units. Dialog units are based on the currently selected font for the dialog and on your screen resolution.

Specify the height of the dialog in dialog units. Dialog units are based on the currently selected font for the dialog and on your screen resolution.

Specify the name of each control on a dialog. This is the name you use in the macro and in Help when referring to this control, so each control on the same dialog should have a unique name. Named Regions are case sensitive.

Display the name of the dialog. This is the name you use in the macro when calling and dismissing this dialog.
Named Regions are case sensitive.

Create an OK button. Clicking this accepts the dialog settings and activates them. Selecting OK fills in the Text box automatically.

Create a Cancel button. Clicking this closes the dialog without implementing the changes. Selecting Cancel fills in the Text box automatically.

Create a Help button. Clicking this button calls the Help file associated with this dialog. Selecting Help fills in the Text box automatically.

Create a blank button and specify any name for it.

Create a button with two possible states (on or off). When users click the button, it does not automatically reflect the change. You must send a message to the dialog within the macro to change the state of the button.

Create a button with two possible states (on or off). When users click the button, it automatically changes state. No additional message within the macro is necessary.

Create a scroll bar that extends the length or width of the control automatically.

Create a scroll bar that appears at the left of the control.

Create a scroll bar that appears at the top of the scroll bar control.

Create a scroll bar that appears at the bottom of the scroll bar control.

Specify the smallest allowable value for scroll bars.

Specify the largest allowable value for scroll bars.

Specify how much each click on the scroll bar will change position.

Specify the initial position of the scroll bar.

Specify which types of files the Select File dialog displays in the List Files of Type combo box. This allows you to limit the types of files that users can choose to list: DOC (document files), SS (spreadsheet files), DB (database files), GR (graphics files), and All Files (*.*) (all files).

Enter the text you want to appear as static text on the dialog. Text longer than the display area is cut off at the right edge.

Select the style of control you want to use (WordPerfect or Windows).

Select the side of the radio button on which the text appears.

Select left, right, or center justification for static text. The text is justified in relation to the text area.

Select the side of the control on which the text will appear.

Create text that appears printed on the dialog without a border around it.

Create text that appears as white text on a darker gray, recessed background.

Create text that appears as black text on a raised, shadowed box.

Create text that appears with ellipses (...) replacing the middle of overly long pathnames.

Specify the name of the variable associated with the dialog control. Use this name in the macro when specifying the values for the variable.

Use this control to specify the path and filename of the file the viewer displays. Click the folder icon to select directories and files.

Display the line of code where the Debugger stopped the macro.

Continue playing the macro until the next defined breakpoint.

Go to the next statement in the macro, play it, then stop.

Go to the return of the next label.

Go to the next line after executing the current one.

Click to define breakpoints for the Debugger inside your macro. You no longer need to use the Step command.

Display all the variables that exist in the current routine. The local variables may have identical names in different user-defined routines. Click the column headings to change the way the variables are sorted. Double-click arrays to expand them. Click the variables to change their contents.

Display the variable type and contents.

Display local variables, which are defined inside the current routine and inside the current macro.

Display global variables, which are defined for all routines in all the macro files that execute with the current macro. Global variables end when the macros finish executing.

Display persistent variables, which are set for all macros running in the current session, independent or interconnected. Persistent variables end when you close the PerfectScript session.

Display and edit the contents of the currently selected variable. If you enclose the value in quotes (""), the variable will be updated with a string value. If you use only numbers without quotes, the variable will be updated with a numeric value.

Update the contents of the selected variable with the contents of the edit box. You must click Update to change the value of a variable. Updating the value of an alias updates the original instance of the variable.

Displays the labels, functions, or procedures that are currently in progress in reverse order with the most recent on top. Select a routine to cause the Variable list box to display the variables that exist in that routine.

Display all the breakpoints defined for the current macro. Click each one to make changes.

Remove all defined breakpoints.

Delete the selected breakpoint.

Disable a breakpoint to let the Debugger ignore it. Enable a breakpoint to return it to activity. Enabled breakpoints are checked, disabled breakpoints are not. You can also use the Spacebar or double-click to enable and disable a breakpoint. You cannot select multiple breakpoints.

Add the currently defined Location, Conditions, and Actions data as a new breakpoint. You must change the breakpoint's data before you can add it.

Update the currently selected breakpoint with the Location, Conditions, and Actions data you added or edited. You must change a breakpoint's data before you can update it.

Specify the type of breakpoint you want to use.

Specify which of the macro files involved in your macro will use this breakpoint. The list may change as the macro runs. You may type in other macro names (without a path). A blank applies the breakpoint to all macros.

Specify how many times (up to 99,999) to let a breakpoint go by before it stops.

Specify the label that the Debugger will stop at. You can also type in a label, with the correct capitalization.

Specify the variable to use as a breakpoint. You can break at every variable in every label, or just the label. Only variables inside the current routine will appear in the list, since each new procedure will have its own pool of local variables. You can also type in variable names, but they must use correct capitalization.

Specify the routine you want to use as a breakpoint.

Specify the .DLL to use as a breakpoint. The macro loads the .DLL, so all currently loaded .DLLs appear in the list. You can also type in a .DLL name (without a path). <all> or a blank uses every .DLL call as a breakpoint.

Specify a particular routine inside the .DLL you chose to use as a breakpoint. You may also type in a routine name, but you must use correct capitalization.

Select the product you want to use as a breakpoint. The list displays all products currently registered with PerfectScript.

Select commands from the list to use as breakpoints. The command list changes, depending on the product you selected.

Specify the line number to use as a breakpoint. You can specify any line number from 1 to 99,999. Each macro file rennumbers the lines, so you can specify a certain line in a certain macro file, or stop at a specified line in all the macro files.

Specify the label to use as a breakpoint. The Debugger stops at the next return, implied or explicit.

Close the dialog box without recording any of the changes you made.

Specify the path and filename of the application where you want to run the macro.

Click the Help button to view an overview topic about this dialog box.

Close the dialog box and implement the changes you made.

Convert a macro into WordPerfect format for compiling and playing.

This is the macro file you selected to play or compile.

Specify the path and filename for the converted macro file. If you do not change the selected path and filename, the converted macro will overwrite the original macro file.

Close the dialog box without saving any of the changes you have made.

Double-click the product command you want to insert into the Command Edit box. Select another type to get a different list of product commands.

Select the parameter you want to insert into the Command Edit text box, then click Edit.

Select the member you want to insert into the Command Edit box, then click Edit.

Select the type of product commands you want to use.

Display the type of parameter currently selected.

Display the value of the currently selected member of a parameter.

Display the token the currently selected command gives to the program.

Double-click commands, then select parameters and enumerations. Click Edit to insert them into this box. Edit the command if necessary, then click Insert to put it into your macro at the insertion point.

Click to insert the currently selected parameter or member into the Command Edit box so you can modify it.

Click to insert the contents of the Command Edit box into your macro at the insertion point.

Automatically open the Debugger when you start a macro.

Automatically open the Debugger when the macro you are playing runs into an error.

Add debugging information to any warnings, errors, or other messages that display while you compile a macro.

Create a file listing each line in your macro with the corresponding line number. This is very useful for debugging macros, since the line numbers correspond to the line numbers the Debugger uses.

Specify the path and filename of the macro editor you want to use. The Editor and the file format you choose must match.

Select the file format your macro editor uses.

Return all General preferences to their original settings.

Specify the directory you want to use as the default macro directory.

Specify the default filename extension you want to use for macros.

Display the macro name, lines compiled, and types of errors while your macro compiles.

This is the macro that is currently compiling.

This is how many lines have been compiled.

These are the number of errors, warnings, and fatal problems your macro contains.

Stop the current compile without finishing it.

Continue the compilation without repairing the error.

This is the problem the compiler ran into. The description tells you what kind of error it is and briefly describes the problem and the line it appears on.

This is the location of the error the compiler ran into. It tells you which macro file to check and which line in the macro has the problem.

Click the format you want to use for recording macros. The format should match the macro editor you select.

Click the number of parameters you want per line.

Select to record all product prefixes when recording macros. The prefixes specify the application the macro should use when you play it.

Select to require named parameters while you record macros.

Specify the maximum length for lines in recorded macros.

Stop the macro you are currently running and debugging.

Click to create and edit breakpoints for the Debugger.

Create a new variable at this point in the macro.

These controls have specific help associated with them.

Expand an array so you can see and edit each variable within it.

This is the name of the macro file where this label, function, or procedure appears.

This is the line number in the displayed macro file where the label, function, or procedure appears. This line number corresponds to the line number in the .WCL file for this macro, but may be different from the line numbering in the original macro file.

Delete the selected variable. If it has content, you can also choose to set the content to undefined instead of deleting the variable entirely.

Click the type of variable you want to create.

Specify a name for the variable.

Specify a macro to compile, then click to compile the selected macro.

Click to add, delete, modify, and change the order of elements in the list.

Type or edit the item you want to add to the list, then click Add to insert the item into the list, or click Replace to change the list item.

These are the items currently in the list. Click items to select them.

Sort the list alphabetically.

Select the item you want to use as the highlighted option in the list when the dialog opens, then click Set Initial.

Insert the item in the List Item text box as a new entry in the list.

Replace the currently selected item with the contents of the List item text box. You can edit list entries by selecting an item, changing it, then clicking Replace.

Remove the selected item from the list.

Move the selected item one place up in the list.

Move the selected item one place down in the list.

This is the list item that will be highlighted when the list opens.

Select the state to determine whether the control will be checked, clear, or grayed when the dialog opens.

Specify the class of the custom control. Standard Windows classes include Button, ComboBox, Edit, ListBox, ScrollBar, and Static. Other controls such as Meters and Spin buttons must be defined in a DLL.

Allow a user to apply attributes (bold, italic, and so on) to text in the edit box.

Allow soft returns in a multiple-line edit box, causing text to wrap to the next line.

Do not allow users to enter tabs in the edit box. Pressing Ctrl+Tab moves the user out of the control.

Create a scroll bar that appears at the right of the control.

Type a value for the parameter, or click Value to allow the application to supply a number.

This is how your macro will appear.

Click to update the displayed macro so that it includes the parameter value you specified.

Apply your current Preferences settings without closing the dialog box.

Use the common Corel suite Open dialog instead of the Windows 95 open dialog.

This is the Toolbar for the PerfectScript Debugger. Use it to quickly access debugging features.


Specify the text for the custom control.

Click the type of measurement the Dialog Editor will use to size and position your dialogs.

Close the dialog on exit.

Set your animation options. Animation allows you to step through macros line by line, automatically bringing up the debugging dialog box at each step so you can check the variables and other calls. You can specify how long the Debugger will pause before moving to the next step. Clicking inside the Debugger halts the animation, as do



any breakpoints you have specified. Click Debug  Animate to resume automatic play.

Type the text for a custom message.

Log custom event messages in the Debugger Event Log.

Remove the variable from the Variables list box.

Remove the contents of the variable but leave the variable in the Variables list box.

Log standard event messages in the Debugger Event Log.

Display the "&" character as itself. If you do not select this option, typing & changes the next letter into a mnemonic.

Specify a style for the custom control. The number in the text box defines a valid style within the control Windows class. For example, an edit control includes styles such as left, multiline, and password. The values for standard Windows styles are in the Windows SDK.

When you use `DialogAddControl` to add a custom control to a dialog box, the `WS_CHILD` (0x40000000) and `WS_VISIBLE` (0x10000000) style bits are added by default.

Put an "X" button on the title bar of the dialog so users can click to close it.

Invoke the Debugger when the macro reaches this breakpoint. If you do not select this option, the Debugger will not break into the macro; the breakpoint will simply be noted as an entry in the Event Log.

Type the comment you want to add to the event log, then click Add. The comment appears next in the event log display and in the .log file, if you save it.

Save the event log as a text file with a filename you specify and a default .log extension.

Remove all entries from the event log.

Connect to the online information and support service you selected.

If you have a modem, these instructions tell you how to use an online service to get more information about suite applications.

Click to configure an online service you selected so you can get more information about suite applications.

Sets the location, size, named regions, variables, and other properties for the custom controls in your dialog box.

Choose a typeface and point size for all text on your dialog. The caption font remains constant for all dialogs. Changes in font size and style affect the size of the dialog.

Specify the sequence in which you want to tab through the controls. You must set the control order before grouping the controls.

Groups controls together. Defining a group box around a set of controls does not group them, and you must set the control order before grouping controls.

Specify the button you want to be activated when the user presses Enter on the control.

Specify the control that has the focus when the dialog opens.

Select the controls you want to tab to.

Enables you to step through code line by line. You can place the cursor over a variable to determine its value.

Generate a list of all macro commands.

List command names and abbreviated parameters.

List command names, descriptions, and parameters.

List command names and parameters.

Exit the macro and close the dialog box.

Open the PerfectScript Debugger and debug the macro.

[View additional information about the macro.](#)

List return values.

